UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Título: Driver para GNU/Linux en espacio de usuario para la cámara QHY5T
Autores: Joaquín Ignacio Bogado García
Director: Lía Hebe Molinari
Codirector:
Asesor profesional:
Carrera: Licenciatura en Informática

Resumen

En las ciencias astronómicas, el uso de cámaras digitales han revolucionado el estudio del Universo. Dado su bajo costo y su rendimiento, muy superior al de la fotografía química, cada vez son más los astrónomos amateurs que las utilizan para retratar los objetos del firmamento.
La cámara QHY5T, diseñada y fabricada por la empresa QhyCCD, es una cámara específica para realizar fotografía lunar, solar y planetaria y para guiado. Cuenta con un puerto de guiado opto acoplado que protege al dispositivo de posibles corrientes parásitas que puedan provenir de los motores de la montura.
Este trabajo trata sobre el desarrollo de un controlador para la cámara QhyCCD QHY5T para el sistema operativo GNU/Linux y de un programa visor (qhy5tviewer) que permite utilizar la cámara desde dicho sistema operativo. Tanto el controlador como el visor se basaron en el driver abierto para GNU/Linux para la cámara QHY5 del mismo fabricante. Ambos controladores fueron desarrollados utilizando la librería libusb y son controladores ``en espacio de usuario", esto quiere decir que en ningún momento el kernel Linux se modifica para que soporte el dispositivo. Muchos de las rutinas de configuración de la cámara de bajo nivel están tomadas del código de controlador para Windows de la cámara QHY5T.

Palabras Claves

GNU/Linux, Driver, Controlador, Cámara QHY5T, Astrofotografía, Fotografía Lunar, Fotografía Planetaria, Fotografía Solar, QhyCCD, qhy5tviewer.

Conclusiones

A comienzos del año 2012 se comprobó que a pesar que existía el soporte para el sensor en el Kernel Linux, la cámara no funcionaba en este sistema operativo. Durante los dos años siguientes se trabajó en el desarrollo de un controlador para GNU/Linux y un programa visor que permitiera grabar imágenes astronómicas, mediante ingeniería inversa, análisis de tráfico y estudio de los controladores de referencia.
Las imágenes presentadas en el capítulo “Conclusiones” fueron obtenidas utilizando exclusivamente GNU/Linux.

Trabajos Realizados

1) Desarrollo de un controlador para GNU/Linux para la cámara QHY5T. Este se basó en el controlador de la cámara QHY5 para GNU/Linux, en el driver para Windows de la cámara QHY5T y en técnicas de ingeniería inversa y análisis de tráfico USB.
2) Desarrollo de un programa de captura o visor para GNU/Linux llamado qhy5tviewer, el cual hace uso del driver comentado en el punto anterior para configurar la cámara, mostrar el flujo de imágenes capturadas y grabarlas en archivos, entre otras funciones.

Trabajos Futuros

1) Agregar soporte para imágenes de 16 bits.
2) Agregar soporte para binning 2x2
3) Migrar el controlador a libusb-1.0
4) Crear el instalador para Debian GNU/Linux
5) Agregar soporte para cambiar parámetros de configuración durante la sesión en curso
6) Mejorar la función write_fits() para incluir metadata

Fecha de la presentación: Abril/2014

Driver para GNU/Linux en espacio de usuario para la cámara QHY5T

Joaquin Bogado

22 de abril de 2014

A mis padres y a mi hermana.

Índice general

Índice de tablas

Índice de figuras

Capítulo 1

Introducción

En las ciencias astronómicas, el uso de cámaras digitales han revolucionado el estudio del Universo. Dado su bajo costo y su rendimiento, muy superior al de la fotografía química, cada vez son más los astrónomos amateurs que las utilizan para retratar los objetos del firmamento, a veces incluso haciendo visible lo que a simple vista no lo es. La astrofotografía, como se conoce a dicha actividad entre los entusiastas, se diferencia de la fotografía tradicional (química o digital) en que las técnicas aplicadas son sustancialmente diferentes y la etapa del procesado puede jugar un papel tan importante como el de la adquisición de los datos. Incluso pueden diferenciarse dentro de la astrofotografía, dos técnicas totalmente distintas, cuyas diferencias debido a que el brillo de los objetos a fotografiar es muy diferente.

Para la fotografía de objetos de espacio profundo, como galaxias y nebulosas, generalmente se realizan algunas decenas de tomas con cierto tiempo de exposición. Estos tiempos pueden variar de segundos a minutos u horas, dependiendo del brillo del objeto y los filtros utilizados, permitiendo que la luz se acumule en el sensor para revelar objetos que no son visibles a simple vista. Esto se debe no solo a que el ojo humano es incapaz de acumular luz como los sensores, sino que tampoco es sensible a muchas longitudes de onda como el infrarrojo o el ultravioleta, en muchas de las cuales emiten algunos objetos.

Cuanto mayor es el tiempo de exposición, mayor es el movimiento aparente del cielo nocturno. Para contrarrestar este movimiento y evitar que las estrellas dejen trazos en las capturas en lugar de aparecer como objetos puntuales, los aficionados utilizan monturas ecuatoriales motorizadas, las cuales una vez puestas en funcionamiento permiten realizar tomas de entre 1 y 3 minutos, dependiendo de la precisión de la puesta en estación[1]. Para tiempos de exposición mayores a los 3 minutos, suele ser necesaria la técnica de "fotografía con guiado" ya que no solo la puesta en estación influye en la calidad del seguimiento celeste, sino también el error periódico introducido por los engranajes de la montura pueden producir variaciones respecto al seguimiento celeste. En esta técnica se utilizan dos telescopios y dos cámaras, una de las cuales debe soportar el protocolo de guiado de la montura que se esté utilizando.

Por otra parte, para la fotografía de objetos brillantes como la Luna, el Sol y los planetas Mercurio, Venus, Marte, Júpiter y Saturno se utiliza otra técnica completamente diferente en la cual se realizan cientos o miles de exposiciones cortas del orden de los milisegundos. En esta técnica, también conocida como Lucky Imaging[1], se descartan aquellas tomas donde la imagen del objeto se encuentra degradada por efecto de la turbulencia atmosférica, procesándose luego solamente aquellos cuadros de "buena calidad".

Existen varios fabricantes de cámaras específicas para estas aplicaciones como SBIG[2], QhyCCD[3] y Orion[4], entre otras. Sin embargo, muy pocas de estas marcas incluyen para sus productos destinados a aficionados, SDKs (Software Development Kits) o drivers para GNU/Linux. SBIG y Apogge[5], solo

[1] La puesta en estación permite contrarrestar el movimiento aparente del cielo realizando un solo movimiento en el eje de Ascensión Recta (AR). Más en
http://www.espacioprofundo.com.ar/verarticulo/Como_alinear_una_montura_ecuatorial_rapidamente.html

[2] http://www.sbig.com/products/cameras/specialty/st-i/

[3] http://qhyccd.com/

[4] http://www.telescope.com/Astrophotography/Astrophotography-Cameras/pc/4/58.uts

[5] http://ccd.com

distribuyen drivers y SDKs para GNU/Linux en su línea de cámaras profesionales, las cuales pueden llegar a costar varios miles de dólares, mucho más de lo que los aficionados promedio argentinos pueden costear.

QhyCCD es una empresa china que desarrolla cámaras para astronomía. Cuenta con varios modelos para los diferentes perfiles de usuario. Las cámaras de guiado son más económicas debido a que generalmente se basan en sensores CMOS (Complementary metal-oxide-semiconductor) en lugar de CCD (Charge-coupled Devices), no poseen buffers de memoria interna y no están refrigeradas. Todos los modelos que ofrece esta empresa cuentan con drivers para las diferentes versiones del sistema operativo Windows. Sin embargo, menos de la mitad de los modelos cuenta con drivers para GNU/Linux, aunque en el último año y medio se ha observado un incremento en interés por parte del fabricante para dar soporte a este sistema operativo. Los primeros drivers para GNU/Linux para las cámaras QhyCCD no fueron distribuidos de manera oficial por la empresa, sino que fueron desarrollados por terceros, con o sin ayuda del fabricante y basándose en técnicas de ingeniería inversa sobre los drivers para Windows de dichas cámaras. Durante el transcurso del 2013 y con la llegada de los nuevos modelos como la QHY5-II, la QHY5L-II y otros, es que el fabricante comenzó a desarrollar un SDK para Linux el cual al momento de escribir este trabajo aun se encuentra en desarrollo y está disponible y en etapa de pruebas[6]. Dicho SDK soporta parcialmente las funciones de los modelos nuevos pero no se tienen noticias acerca de si se agregará soporte para los modelos anteriores a 2013.

Existen tres modelos de cámaras de guiado QhyCCD de la serie QHY5

- El modelo QHY5 se basa en el sensor MT9M001[2] fabricado por Aptina (antes Micron) y tiene dos versiones, color y mono. Es la única cámara de la serie que cuenta con un controlador para GNU/Linux desarrollado en 2009 por Geoffrey Hausheer. Este driver es distribuido con una licencia GPL2[7].
- El modelo QHY5V está basado en otro sensor del mismo fabricante. No se conocen drivers para GNU/Linux para este modelo al momento de escribir este trabajo.
- El modelo QHY5T basado en el sensor MT9T001[3] también fabricado por Aptina, tiene un sensor más grande que los modelos anteriores y solo está disponible en formato color.

El driver desarrollado para este trabajo es para el modelo QHY5T y se basó en el driver abierto para el modelo QHY5. A pesar de que los tres modelos son muy similares en cuanto a funciones, las tres cámaras son radicalmente diferentes internamente (solo el tamaño del sensor, diferente en cada modelo, cambia radicalmente el funcionamiento de ambas cámaras), por lo que las modificaciones realizadas al driver existente para el modelo QHY5 fueron sustanciales. Ambos controladores fueron desarrollados utilizando la librería libusb y son controladores "en espacio de usuario", esto es, en ningún momento el kernel Linux se modifica para que soporte el dispositivo. Esto tiene algunas ventajas y desventajas que se discuten en el capítulo 6.

El trabajo realizado para esta tesina consistió en:

- La extracción del firmware de la cámara mediante técnicas de análisis de tráfico USB.
- El desarrollo de un controlador para GNU/Linux para el modelo de cámara QHY5T, el cual se baso en parte en el trabajo realizado por Hausheer para el controlador de la cámara QHY5, en el driver para Windows de la cámara QHY5T desarrollado por Tom Van den Eede y en técnicas de ingeniería inversa y análisis de tráfico USB.
- El desarrollo de un programa de captura o visor para GNU/Linux llamado *qhy5tviewer*, el cual hace uso del driver comentado en el punto anterior para configurar la cámara, mostrar el flujo de imágenes capturadas y grabarlas en archivos, entre otras funciones.

En el capítulo 2 se describe el hardware de manera detallada, haciendo énfasis en las características principales del sensor de la cámara. En el capítulo 3 se describe el proceso por el cual se obtuvo una de las versiones del firmware mediante técnicas de ingeniería inversa. En el capítulo 4 se describen detalladamente las rutinas implementadas que permiten acceder a las funciones de la cámara y las

[6] La última versión de dicho SDK se puede descargar de https://github.com/qhyccd-lzr/QHYCCD_Linux. El autor de esta tesina cuenta con algunas contribuciones de código a dicho proyecto.

[7] La licencia General Public License versión 2 es una licencia de software libre avalada por la Free Software Foundation

cuales forman parte de la API del driver, además de las rutinas internas de comunicación y sincronización privadas del driver. En el capítulo 5 se describe la aplicación desarrollada para la captura de imágenes utilizando el dispositivo, la cual hace uso intensivo de la API descripta en 4. En el capítulo 6 se detallan los objetivos alcanzados, se muestran algunos resultados obtenidos y se presentan algunas conclusiones. Por último capítulo 7 se discuten posibles mejoras tanto al driver como al visor. Además se proveen instrucciones precisas sobre como obtener el código del controlador y del visor, además de instrucciones para la compilación y puesta en funcionamiento.

Capítulo 2

El hardware

La QHY5T, diseñada y fabricada por Hongyun Qiu para su empresa QhyCCD, es una cámara para realizar fotografía lunar y planetaria y para guiado. Cuenta con un puerto de guiado opto acoplado que protege al dispositivo de posibles corrientes parásitas que puedan provenir de los motores de la montura. Tiene un sensor CMOS de 3 Megapixels (Mpx) color, fabricado por Aptina modelo MT9T001[3]. El procesador de la cámara está basado en la arquitectura 8051 de Intel y es fabricado por Cypress[4]. La cámara no cuenta con memoria interna para almacenar las fotografías. Estas se transmiten a la computadora a través del puerto USB de manera inmediata.

En Windows, cuando la cámara se conecta, el driver se encarga de copiar el firmware a la cámara. Este firmware no queda residente en la cámara y debe cargarse nuevamente cada vez que la cámara se desconecta y se vuelve a conectar al puerto USB de la computadora. Una vez cargado el firmware, el driver envía los parámetros de configuración a la cámara a través de la conexión USB. Entre los parámetros de configuración, se pueden destacar la resolución, la ganancia del sensor (gain), el tiempo de exposición, el binning y la cantidad de bits por pixel de la imagen devuelta por la cámara. Luego el firmware procesa los requerimientos de nuevos frames[1] del driver. El procesador de la cámara hace de interfaz entre el sensor y la computadora.

2.1. El sensor

El sensor de imagen es el elemento de una cámara fotográfica digital que capta la luz que compone la fotografía. Se trata de un chip formado por millones de componentes sensibles a la luz que al ser expuestos capturan la imagen. Cada uno de los elementos fotosensibles del sensor se denomina pixel (picture element). El número de píxeles del sensor se suele medir en millones de píxeles (o megapíxeles, Mpx). Dependiendo de la tecnología utilizada en su fabricación pueden dividirse a grandes rasgos en sensores CCD o CMOS, siendo los últimos más económicos y eficientes en cuanto a consumo de energía, entre otras ventajas.

Una característica importante a mencionar es que dado que los foto-receptores de silicio son incapaces de determinar la longitud de onda de los fotones recibidos, son inherentemente monocromáticos. Los sensores color utilizan un Color Filter Array (CFA), es decir una matriz de filtros rojos, verdes y azules con un patrón conocido como *Bayer matrix* o *Bayer pattern*[5]. La imagen en color es calculada mediante un proceso de interpolación conocido como interpolación cromática, debayerización[2] o *bayer demosaicing*[6].

La QHY5T tiene un sensor CMOS color de 3Mpx modelo MT9T001 fabricado por Aptina. Este sensor cuenta con una matriz de pixels de 2112 columnas por 1568 filas como se muestra en la figura 2.1. Las columnas 0 a 27 y 2085 a 2111 y las filas 0 a 15 y 1561 a 1567 devuelven información del negro óptico, es decir, no están expuestas a la luz. Esta información se utiliza de manera interna por la lógica del sensor para calibrar el nivel de negro (*black level* en la jerga) de la imagen, aunque también pueden ser leídos configurando dos registros a través de la interfaz I^2C[3]. Además, hay 2057 columnas

[1] En este trabajo, se utilizará frame, cuadro o fotograma de manera indistinta, para hacer referencia a una matriz de datos que representa una imagen capturada por el sensor de la cámara.

[2] Se trata el problema de la debayerización en la sección 5.5

[3] Inter-Integrated Circuit bus

y 1545 filas de pixels activos que proveen un borde de 4+ pixels alrededor de la imagen de 2048x1536, lo que deja espacio para el tratamiento de bordes durante la interpolación para generar la imagen en color.

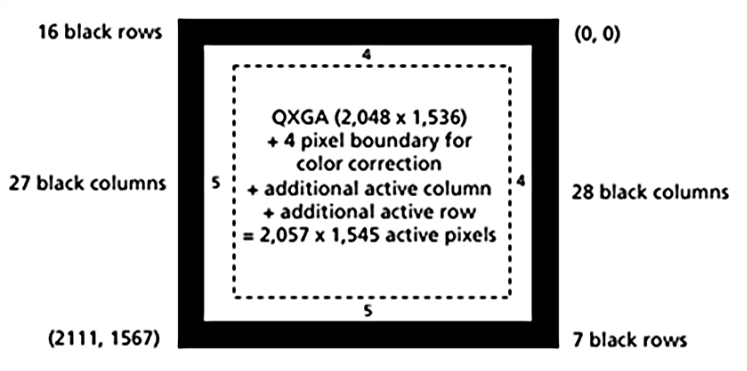


Figura 2.1: Estructura del sensor MT9T001

El sensor MT9T001 tiene un patrón de Bayer como se muestra en la figura 2.2. Las filas pares tienen pixels rojos y verdes, mientras que las impares tienen pixels verdes y azules.

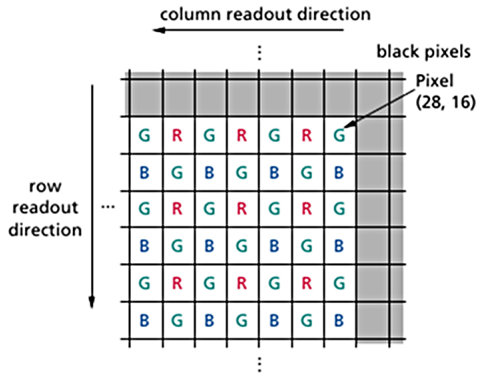


Figura 2.2: Patrón de Bayer del sensor MT9T001

La señal analógica de la matriz de pixels es primero amplificada en la etapa de ganancia programable. Luego esta señal pasa a la circuitería de corrección de desplazamiento analógica, donde es posible configurar desplazamientos para cada canal por separado. Esto permite por ejemplo que el usuario corrija artefactos de canales desalineados por la introducción de lentes u otros dispositivos en el tren óptico. Luego la señal analógica es cuantificada y muestreada en una unidad ADC [4] de 10 bits. Posteriormente se aplican las transformaciones digitales (gain y offset) para finalmente hacer disponibles los datos a través de los pines $D_{out}[9:0]$.

[4] Analog-to-Digital Converter

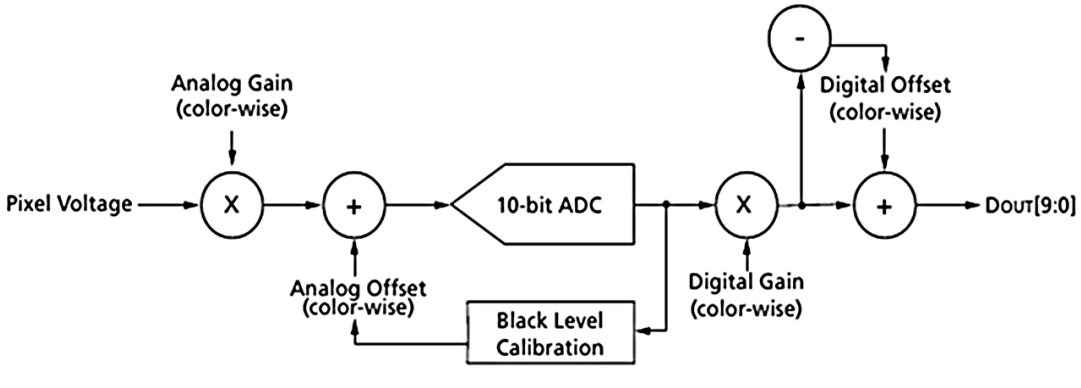


Figura 2.3: Camino de la señal analógica

Este sensor tiene una interfaz nativa I^2C. Todas las configuraciones posibles para el sensor se logran escribiendo los valores adecuados en registros como los detallados en la tabla. Sin embargo, el firmware de la cámara no expone esta interfaz sino que implementa una capa de abstracción que impide acceder a las funciones nativas I^2C del sensor.

La tabla 2.1 resume algunos de los registros más importantes. La lista completa puede consultarse en la tabla 3 de la página 12 de la hoja de datos del sensor[3].

Registro	Descripción	Formato	Default
0x01	Fila de comienzo	0000 00dd dddd ddd0	0x0014
0x02	Columna de comienzo	0000 0ddd dddd ddd0	0x0020
0x03	Cantidad de filas	0000 00dd dddd ddd1	0x05FF
0x04	Cantidad de columnas	0000 0ddd dddd ddd1	0x07FF
0x05	Horizontal blanking	0000 00dd dddd dddd	0x008E
0x06	Vertical blanking	0000 00dd dddd dddd	0x0019
0x22	Modo de lectura para filas	0ddd 0ddd 0ddd 0ddd	0x0000
0x23	Modo de lectura para columnas	0000 0ddd 00dd 0ddd	0x0000
0x2B	Gain para Verde1 (G1)	0ddd dddd 0d0d dddd	0x0008
0x2C	Gain para Rojo (R)	0ddd dddd 0d0d dddd	0x0008
0x2D	Gain para Azul (B)	0ddd dddd 0d0d dddd	0x0008
0x2E	Gain para Verde2 (G2)	0ddd dddd 0d0d dddd	0x0008
0x35	Gain global	dddd dddd dddd dddd	0x0008

Tabla 2.1: Resumen de los registros más importantes del sensor y sus valores por defecto.
NOTA: `0` = bit siempre en cero; `1` = bit siempre en uno; `d` = bit programable

Descripción de las opciones de configuración

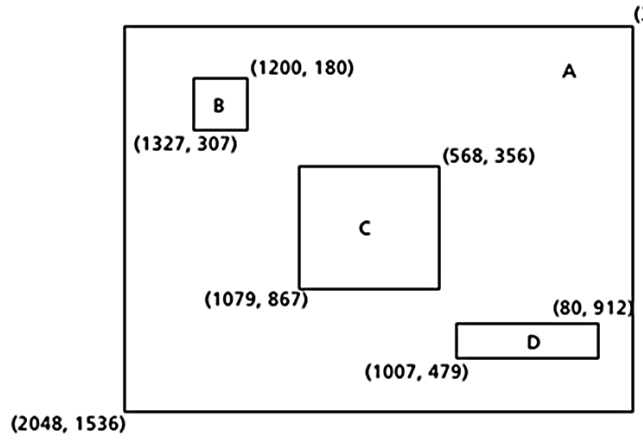
Fila de comienzo (Reg 0x01):
Indica cual es la primera fila a ser leída. Por defecto es 0x14, es decir que la primera fila a ser leída es la 20. Debe ser un número par.

Columna de comienzo (Reg 0x02):
Indica cual es la primera columna a ser leída. Por defecto es 0x20, es decir que la primera columna a ser leída es la 32. Debe ser un número par.

Cantidad de filas (Reg 0x03):
Indica alto de la frame (cantidad de filas - 1 del frame). Por defecto es 0x5FF, es decir que el frame por defecto es de 1536 filas. Debe ser un número impar.

Cantidad de columnas (Reg 0x04):
Indica ancho de la frame (cantidad de columnas - 1 del frame). Por defecto es 0x7FF, es decir que el frame por defecto es de 2048 columnas. Debe ser un número impar.

Utilizando los cuatro registros antes mencionados es posible declarar regiones de interés (ROIs) de manera arbitraria en múltiplos de a 4 pixels. Una ROI queda definida mediante la especificación de una resolución y un desplazamiento respecto al origen en los ejes x e y. Para este sensor, es posible definir ROIs por hardware, es decir que una vez configurada una ROI, el sensor solo devuelve los datos para esa zona y no es necesario recortar los datos luego. Este comportamiento trae aparejado un aumento en la eficiencia, ya que no consume el mismo tiempo enviar los datos correspondientes a una imagen de 640x480 pixels a través de la interfaz USB, que enviar los datos de una imagen de 2048x1536 pixels para luego recortar la región de interés por software. Debido a esto, se pueden lograr un frame rate mayor a resoluciones bajas. Cabe resaltar que el uso de salteos de lectura (*Row Skip* y *Column Skip*) y el uso de *binning*[5] pueden determinar algunos límites para los valores de estos registros.



	Window Size	Reg0x01	Reg0x02	Reg0x03	Reg0x04
Window A	2,048 x 1,536	0x0014	0x0020	0x05FF	0x07FF
Window B	128 x 128	0x00B4	0x04B0	0x007F	0x007F
Window C	512 x 512	0x0164	0x0238	0x01FF	0x01FF
Window D	400 x 96	0x0390	0x0050	0x018F	0x005F

Figura 2.4: Ejemplos de diferentes ROIs

Blanking control (Reg 0x05 y 0x06):
Estos registros controlan el tiempo de *blanking* entre filas (*Horizontal blanking*) y entre frames (*Vertical blanking*). El *blanking* horizontal está dado en tiempos de pixel (pixel clocks) y el *blanking* vertical en tiempos de lectura entre filas (*row readout time*).

Modo de lectura para filas (Reg 0x22):
Este registro se divide en dos. Los bits `[2:0]` permiten configurar el *Row Skip*. Por ejemplo, 0b000 indica leer todos los pares de filas (no skip), 0b001 indica leer dos filas y saltarse dos filas (skip 2x), es decir, leer las filas 0, 1, 4, 5, 8, 9, etc, 0b010 indica leer dos filas y saltarse cuatro (skip 3x), es decir, leer las filas 0, 1, 6, 7, 12, 13, etc y así siguiendo.

Los bits `[5:4]` permiten configurar el binning para las filas. Se admiten tres modos de binning para filas. Si el registro vale 0 no hay binning (también se conoce como binning 1x). Si el registro vale 1, se emplea para las filas el binning 2x. Esto se logra promediando dos pixels adyacentes (del mismo color). Si el registro vale 2 se utiliza binning 3x, esto es, el promedio de 3 pixels adyacentes del mismo color.

[5]El binning es una forma de cuantización. En este caso, se utilizan los datos adquiridos de varios foto-receptores contiguos para determinar el valor para un pixel.

Modo de lectura para columnas (Reg 0x23):
Idem que el punto anterior. Notar que el tanto el binning como el skip pueden aplicarse a filas y columnas de manera independiente. Sin embargo, lo más común es utilizar binning 2x2 o 3x3, es decir, binnings cuadrados. La utilización del binning mejora el rendimiento del sensor respecto al ruido de lectura, aunque reduce la resolución. La utilización de salteo de filas y columnas permite reducir la resolución y el tráfico de datos sin que se vea reducido el campo de visión[6].

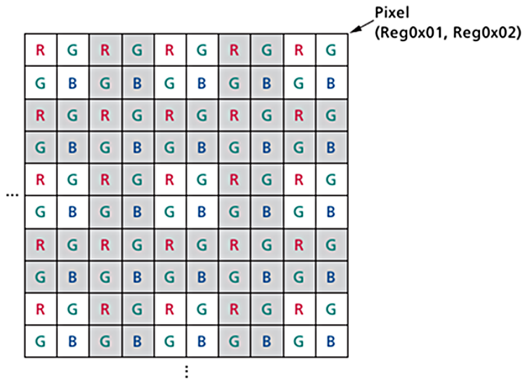


Figura 2.5: Utilización de salteo de lectura 2x2 (row skip 2x, column skip 2x)

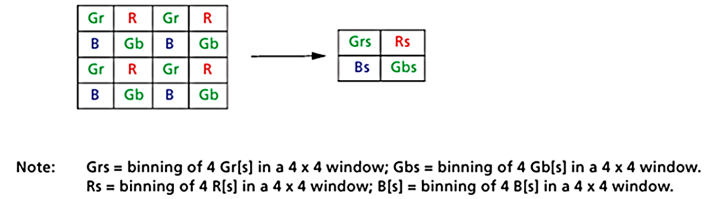


Figura 2.6: Binning 2x2

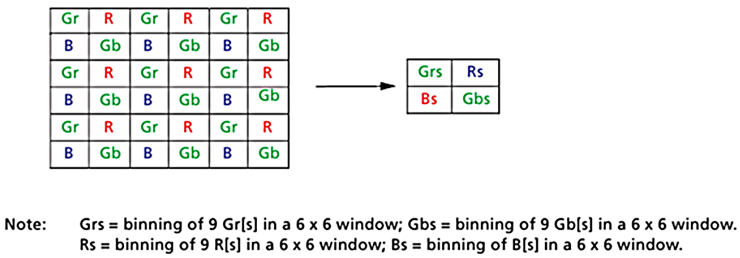


Figura 2.7: Binning 3x3

Ganancia por canal (Reg 0x2B, 0x2C, 0x2D y 0x2E):
Estos registros también están divididos en dos. Los bits `[6:0]` controlan la ganancia analógica mientras que los bits `[14:8]` controlar la ganancia digital. La etapa programable de ganancia analógica consiste en dos etapas diferentes que trabajan como un pipeline. La primer etapa tiene un rango para la ganancia de 1 a 2 y es controlada por el bit `[6]`. La segunda etapa controlada por los bits `[5:0]` tienen un rango para la ganancia de 1 a 4 en incrementos de a 0.125. La ganancia se puede configurar independientemente para cada color del patrón de Bayer y se puede calcular utilizando la siguiente fórmula:

[6]Field of View o FOV, en la jerga

$GAIN_{total} = GAIN_{analog} * GAIN_{digital}$
$GAIN_{analog} = (1 + Bit[6]) * (Bit[5:0] * 0{,}125)$
$GAIN_{digital} = 1 + Bit[14:8]/8$

De esta manera el rango total para la ganancia es de 1 a 128. Dado que el bit `[6]` es un factor multiplicativo para la configuración de la ganancia, es posible obtener el mismo total nominal para la ganancia con más de una configuración. Sin embargo algunas configuraciones son mejores ya que presentan menos ruido de lectura. En el datasheet del sensor[3] puede encontrarse la siguiente tabla de valores recomendados para los diferentes valores de ganancia.

Ganancia	Incremento	Valor recomendado
1 - 4	0.125	0x0008 - 0x0020
4.25 - 8	0.25	0x0051 - 0x0060
9 - 128	1	0x0160 - 0x7860

Tabla 2.2: Valores óptimos para los niveles de ganancia en relación al ruido.

También es posible configurar la ganancia global utilizando Reg0x35. Sin embargo este comportamiento no es abstraído por el firmware con lo que queda inaccesible desde las funciones del driver. Notar que el rango de 1 a 128 no es lineal. En el driver este rango es de 1 a 167 como consecuencia de transformar la función de ganancia en una función lineal a pasos enteros. Se discute más al respecto en la sección 4.9.

2.2. El procesador

La cámara cuenta con un procesador modelo CY7C68013A fabricado por Cypress Semiconductor Corporation. Dicho procesador cuenta con una serie de características que se enumeran a continuación:

- Reloj del sistema de 24 MHz.
- Set de instrucciones basado en la arquitectura Intel 8051, con algunas instrucciones y registros añadidos.
- Cuatro Endpoints USB programables para transferencias *BULK*, *INTERRUPT* o *ISOCHRONOUS*[7] con capacidades de doble, triple y cuádruple *buffer*.
- Transceptor[8] USB 2.0 de alta velocidad integrado.
- Soporte de compatibilidad con EZ-USB FX2[9].
- En el caso de la cámara QHY5T, el código del firmware en el lenguaje de la arquitectura 8051 corre desde la memoria RAM interna del procesador, la cual es cargada por USB (mediante el protocolo FX2) al momento en que la cámara es conectada a la computadora.
- Memoria RAM integrada de 16 KB para datos y/o instrucciones.
- Interfaz de datos de 8 o 16 bits con conversión automática.
- Interfaz I^2C (solo en modo Master) a 100/400 KHz.

Este procesador ejecuta un firmware que atiende los requerimientos del driver. Los requerimientos (que pueden ser mensajes de control o requerimiento de datos del sensor) son recibidos por la interfaz USB. Estos requerimientos son luego transformados en mensajes para el sensor o en pulsos para el puerto de guiado según corresponda.

Debido a esto, la interfaz I^2C del sensor queda oculta e inaccesible al controlador y no es posible comunicarse con el primero sino a través de las funciones implementadas en el firmware.

[7] Estos tipos de transferencia son los soportados por el protocolo USB. Más en http://www.beyondlogic.org/usbnutshell/usb4.shtml

[8] Dispositivo transmisor y receptor.

[9] Un protocolo diseñado por AnchorChips que permite cargar el firmware al procesador por USB, ya que el protocolo USB está soportado en forma nativa en el dispositivo. Más en http://www.linux-usb.org/ezusb/

2.3. El puerto de guiado

La cámara cuenta con una interfaz de guiado que permiten controlar una montura que soporte el protocolo ST-4 el cual se describe en [7]. Dicha interfaz conecta la cámara a la montura a través de un cable de 6 hilos con un conector RJ-25 en cada uno de sus extremos. El conector RJ-25 tiene el mismo formato que el RJ-11 o el RJ-14, utilizados comúnmente en telefonía, pero cuenta con 6 pines en lugar de 2 o 4 respectivamente. Además, el cableado utilizado para el protocolo ST-4 es diferente.

A través de el puerto de guiado es posible enviar pulsos de guiado a la montura, en las cuatro direcciones posibles que una montura ecuatorial admite. Estas son Ascensión Recta (RA por sus siglas en inglés) positiva y negativa y Declinación (DEC) positiva y negativa.

- Pin 1: No conectado
- Pin 2: Conectado a Ground
- Pin 3: Pulso en RA+
- Pin 4: Pulso en DEC+
- Pin 5: Pulso en DEC-
- Pin 6: Pulso en RA-

Dado que este puerto se conecta directamente a la lógica de la montura y que la especificación del protocolo ST-4 no indica como se implementa esta conexión, los circuitos de la cámara se encuentran aislados mediante opto acopladores. Esto protege a la cámara de monturas que conectan dicho puerto directamente al circuito de potencia de los motores, los cuales por su naturaleza pueden producir corrientes parásitas de varias veces el voltaje soportado por la electrónica de la cámara y que la pueden dañar permanentemente.

El protocolo ST-4 es un estándar *de facto*, introducido en el mercado por el fabricante SBIG para su cámara guía ST-4 Star Tracker Imaging Camera [7]. Esta cámara ha sido la referencia tanto para fabricantes de otras cámaras de guiado como también para fabricantes de monturas, los cuales dan soporte por lo menos a este protocolo o alguna modificación compatible. Sin embargo no existe una especificación oficial que describa por ejemplo que voltajes debe manejar cada línea (aunque generalmente es el voltaje manejado por la cámara internamente) y los pulsos de guiado se activan por flancos.

A diferencia de la cámara de SBIG, la QHY5T no dispone de hardware especializado para el análisis de las imágenes en pos de determinar el desplazamiento de las estrellas. Esta tarea debe hacerse por software en la computadora, mediante algún programa especializado.

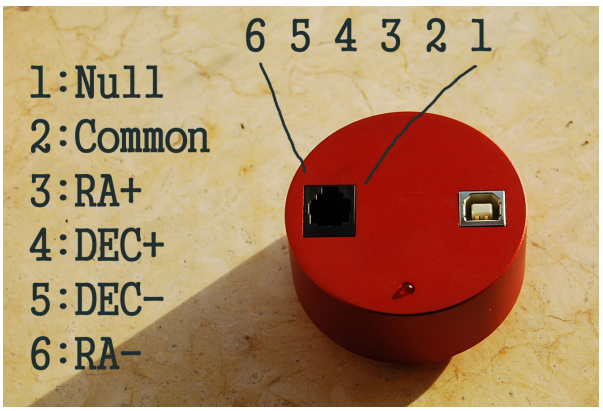


Figura 2.8: Puerto de guiado en la cámara QHY5T

Figura 2.9: Interfaz de conexión de una montura HEQ5 marca SkyWatcher. A la derecha, el puerto de guiado compatible con el protocolo ST-4.

Capítulo 3

El firmware

Durante el transcurso de este trabajo se obtuvieron dos versiones del firmware por medios totalmente diferentes. La primera versión fue obtenida por medio de ingeniería inversa y análisis de capturas de tráfico USB de la comunicación entre el driver y el dispositivo en un entorno con sistema operativo Windows, mientras que la segunda versión fue provista por el fabricante en una etapa bastante avanzada del presente trabajo. Dados los buenos resultados obtenidos mediante la aplicación de la primer técnica es que se describe su obtención por esta vía.

A pesar de los buenos resultados, el proceso es sumamente largo y en algunas partes, tedioso. Requirió mucho tiempo, esfuerzo, lectura, análisis y sobre todo, paciencia.

3.1. Las primeras capturas

Pocos días después de la adquisición del dispositivo y tras constatar que efectivamente la cámara no funcionaba en el sistema operativo GNU/Linux, se decidió comenzar a trabajar en hacer ingeniería inversa del controlador de la cámara. La idea consistía en capturar el tráfico USB que enviaba el controlador al dispositivo y duplicarlo en un programa que utilice la librería libusb, de manera similar a lo que se hacía en el controlador de la cámara QHY5.

Los primeros intentos de capturar el tráfico USB se realizaron utilizando una máquina virtual QEMU+KVM ejecutando Windows XP SP3 y corriendo sobre una máquina física Lenovo Thinkpad R400 con 4GB de memoria RAM ejecutando Gentoo GNU/Linux. Estos intentos fueron infructuosos debido a que la cámara no funcionaba en el entorno propuesto, probablemente debido a que la cámara requiere una conexión USB de alta velocidad con soporte completo para la especificación USB 2.0. Posteriormente se probaron otras plataformas virtuales como VMWare y VirtualBox con idénticos resultados. La cámara no funcionaba sobre Windows en una máquina virtual.

Las primeras capturas efectivas se obtuvieron utilizando una máquina Lenovo Ideapad S10 ejecutando una versión de Windows 7 Starter Edition, en la cual se instalaron el controlador de la cámara para dicha versión del Sistema Operativo, el programa de captura QGVideo5T[1] para la cámara QHY5T y el programa USBlyzer[2] para captura, análisis y reporte de tráfico USB. Tanto el controlador para Windows como el programa de captura QGVideo5T fueron provistos por el fabricante a través de la página del producto. Para las capturas se utilizó la versión de pruebas de USBlyzer, cuyas funciones alcanzaron para realizar los reportes.

Se notó que al conectar la cámara, el led de la misma parpadeaba durante unos segundos para luego quedar permanentemente prendido hasta que comenzara la captura de video (luego parpadeaba a la velocidad de captura). Esto indicaba que el tráfico USB comenzaba ni bien se conectaba la cámara y hacía imprescindible comenzar a capturar el tráfico antes de conectar la cámara al puerto USB.

Los resultados esperados de este proceso eran poder identificar aquellos paquetes del tráfico que contuvieran parámetros de configuración, como tamaño y posición de la ventana de captura, ganancia o tiempo de exposición. Muchos de estos parámetros tenían valores por defecto conocidos y se podían modificar a través del programa de captura QGVideo5T. Además también se esperaba poder aislar

[1] El programa puede descargarse de http://qhyccd.com/file/repository/QHY5T/QGV5TINST1_0.zip
[2] http://www.usblyzer.com/

estos paquetes de los paquetes de datos de los frames capturados por el sensor. Se presumía que una vez enviado el tráfico de configuración, la cámara tomaría todos los frames con esa misma configuración y no se volvería a enviar información de control hasta que se realice un cambio en la configuración a través del programa de captura de video. Esto se pudo constatar simplemente cubriendo el sensor de manera que no entre luz a la cámara, lo que resultó difícil ya que por más que la cámara estuviera cubierta la imagen del video no era negra, sino gris, lo que indicaba que los bytes leídos del sensor no eran 0x00 y podían confundirse con datos de control. Por el contrario, exponiendo el sensor a plena luz, todos los bytes leídos del sensor eran 0xFF, es decir pixels totalmente blancos.

Se pudo constatar de esta manera lo que se presumía en una primera instancia, es decir, no se transfiere información de control entre el controlador y la cámara a menos que alguna configuración cambie. Sin embargo se notó un comportamiento no esperado cuando se iniciaba la captura antes de conectar la cámara al puerto USB. Al momento de capturar antes de conectar la cámara, se podía apreciar un flujo de tráfico el cual no se podía asociar ni a datos del sensor ni a parámetros de configuración. Este flujo de datos duraba solo uno o dos segundos, luego de los cuales la conexión se cortaba y la captura se detenía. Era necesario reiniciar la captura con la cámara conectada el puerto USB para poder capturar el tráfico de configuración o de datos.

Compartiendo información del proyecto con un colega, se pudo determinar que en la mayoría de las cámaras web o similares es responsabilidad del driver subir el firmware al dispositivo, dado que en general no tienen una memoria EEPROM o similar para almacenamiento no volátil y se comenzó a sospechar que dicho flujo de datos anormal podía contener dicho firmware.

Con esto en mente se comenzó a trabajar en separar el código del firmware de los reportes de captura provistos por el USBlyzer.

Como se pudo determinar luego, otras cámaras del mismo fabricante se comportaban de manera similar, es decir, el controlador graba el firmware del dispositivo al momento de la conexión en una memoria volátil, por lo que es necesario realizar esta tarea cada vez que el dispositivo se conecta. Además la versión para GNU/Linux del driver para la cámara QHY5 solucionaba esta tarea utilizando el mecanismo *udev* para detección de dispositivos de GNU/Linux y *fxload*, el cual permite cargar el firmware en formato IHEX a la memoria del microcontrolador de la cámara. Es decir, el firmware no formaba parte del código fuente del driver sino que estaba separado y por ello no fue advertido en un primer análisis de dicho código fuente.

Dicho esto, ahora había pues que convertir el reporte obtenido de la captura con flujo anormal a un archivo IHEX compatible con *fxload*.

3.2. Extracción de datos de la captura de tráfico USB

La versión utilizada del programa de captura de tráfico USB genera un reporte en formato html en una tabla con la siguiente estructura.

```
<table>
<tr>
<th>Type</th>
<th>Seq</th>
<th>Time</th>
<th>Elapsed</th>
<th>Duration</th>
<th>Request</th>
<th>Request Details</th>
<th>Raw Data</th>
<th>I/O</th>
<th>C:I:E</th>
<th>Device Object</th>
<th>Device Name</th>
<th>Driver Name</th>
<th>IRP</th>
<th>Status</th>
</tr>
⋮
```

De estas columnas la más importante es la columna *Raw Data* la cual contiene los datos crudos de la transferencia de datos. Utilizando expresiones regulares se obtuvieron los datos de dicha columna

los cuales están formados por 8 dígitos hexadecimales para dirección, dos espacios en blanco, 16 bytes hexadecimales separados por espacios en blanco, dos espacios en blanco y la representación ASCII de los 16 bytes precedentes. Por ejemplo:

```
00000000  12 01 00 02 FF FF FF 40 18 16 10 09 00 00 00 00  .......@........
00000010  00 01
00000000  12 01 00 02 FF FF FF 40 18 16 10 09 00 00 00 00  .......@........
00000010  00 01                                           ..
00000000  12 01 00 02 FF FF FF 40 18 16 10 09 00 00 00 00  .......@........
00000010  00 01                                           ..
00000000  01                                              .
00000000  01                                              .
00000000  01                                              .
00000000  02 02 EE 32                                     ...2
```

⋮

Debido a una incorrecta configuración de los filtros del programa de captura de tráfico, los datos crudos aparecían triplicados y el campo wvalue de los paquetes USB no figuraba en los reportes. Los datos triplicados fueron borrados a mano con un editor de texto y la información del campo wvalue fue copiada a mano usando lápiz y papel, para su posterior tratamiento. Además se eliminaron la primer línea del archivo generado debido a que correspondía a una transferencia de datos de entrada, las direcciones y la representación ASCII.

El archivo generado se adjunta en forma separada con el nombre de **q.hex**, el cual a pesar de tener extensión .hex no es un archivo IHEX bien formado.

3.3. El formato IHEX

El formato de archivo IHEX (Intel HEXadecimal) es un formato de archivos para representación de archivos binarios en texto plano, cuyas direcciones son absolutas y esta orientado a la programación de microcontroladores. Los datos binarios son representados en BCH (dos dígitos hexadecimales por cada byte de datos) y agrupados en registros (un registro por cada línea). Cada registro está formado por:

1. **Record Mark:** El carácter ASCII ":" que marca el inicio de un registro.
2. **Data lenght:** 1 par de dígitos hexadecimales que indican la longitud del campo de datos del registro.
3. **Load offset:** 2 pares de dígitos hexadecimales en big endian que indican la dirección donde se cargarán los datos. Solo tiene sentido si es un registro de datos. Si no debe ser "0000"
4. **Record type:** 1 par de dígitos hexadecimales que indican el tipo de registro. Pueden ser desde 00 hasta 05.
5. **Data:** n pares de dígitos hexadecimales que representan los datos.
6. **Checksum:** 1 par de dígitos hexadecimales con el byte menos significativo del complemento a dos de la suma de todos los dígitos hexadecimales de los campos 2 al 5.

Por otra parte, los tipos de registro pueden ser:

- **00**: Registro de datos
- **01**: Registro de fin de archivo
- **02**: Registro extendido de segmento de dirección
- **03**: Registro de comienzo de segmento de dirección
- **04**: Registro extendido de dirección lineal

- **05**: Registro de comienzo de dirección lineal

Para este trabajo solo fueron necesarios los registros de tipo 00 y 01.

De las capturas de tráfico USB se contaba con el campo de datos para armar el archivo en formato IHEX pero no con las direcciones. Estas direcciones fueron documentadas en lápiz y papel inspeccionando el campo *wvalue* de los paquetes de datos USB con la herramienta USBlyzer. La mayoría de las direcciones correspondían a escrituras secuenciales donde los datos eran escritos en registros consecutivos. Además, era necesario calcular los *checksums* para cada uno de los registros. Fue necesario entonces escribir un programa que transforme el archivo **q.hex** en un archivo IHEX bien formado.

Dicho programa se adjunta con el nombre **qtohex.c** cuya entrada es el archivo **q.hex** y la salida el archivo **qhy5t.hex**. Cuenta con tres funciones, `readline()`, `checksum()` y `write_record()`, además de `main()`, por supuesto. La función `readline()` devuelve un arreglo de caracteres con una línea del archivo q.hex sin espacios, además de la longitud de dicho arreglo. La función `write_record()` (figura 3.2) recibe como parámetros un puntero al archivo de salida, la longitud del arreglo de datos, la dirección de los datos, el tipo de registro (siempre 00, indicando que es un registro de datos salvo el último registro del archivo, cuyo tipo es 01) y el arreglo de datos y escribe el registro en el archivo de salida incluyendo el carácter de comienzo de registro `:` y el checksum, el cual se obtiene invocando a la función (figura `checksum()`3.3) con el arreglo de datos y su longitud.

Dado que los desplazamientos del registro *wvalue* no habían sido extraídos en forma programática, en main se hicieron los llamados a `write_record()` hardcodeando las direcciones en cada llamada, excepto cuando las escrituras eran a direcciones consecutivas.

El siguiente fragmento de código3.1 muestra como se realizan las llamadas a `write_record()`. Notar el `while`, el cual se repite 23 veces y en el cual se escriben registros en direcciones consecutivas. En `main` hay unos 11 bloques de direcciones como este y unos 10 donde las direcciones son aleatorias y fue necesario especificarlas una a una.

```
int main(){
...
  len = readline(cmd, fin);
  write_record(fout, len, 0xE600, 0, cmd);
  len = readline(cmd, fin);
  write_record(fout, len, 0x0200, 0, cmd);
  wvalue=0x020B;
  while (wvalue <= 0x02EB){
    len = readline(cmd, fin);
    write_record(fout, len, wvalue, 0, cmd);
    wvalue+=8;
  }
...
```

Figura 3.1: Fragmento de la función main()

El resultado de este proceso es el archivo **qhy5t.hex** el cual se adjunta a este trabajo, el que además de ser un IHEX bien formado, contiene la primer versión del firmware para la cámara QHY5T obtenida mediante el análisis de tráfico USB. Como se pudo determinar luego, esta versión del firmware solo difiere de la liberada por el fabricante en el primer registro, el cual está de más en la versión presentada en este trabajo y corresponde a una señal de reset que se envía al microprocesador de la cámara para indicar que a continuación se cargará el firmware. A pesar de esto, es posible cargar el firmware obtenido por análisis de tráfico USB a la cámara mediante la utilización del programa *fxload* y el funcionamiento del dispositivo es exactamente el mismo con cualquiera de las dos versiones del firmware.

```
write_record(FILE * fout, int len, int offset, int type, char * cmd){
  char b[256];
  sprintf(b, ":");
  sprintf(b+1, "%02X", len);
  sprintf(b+3, "%04X", offset);
  sprintf(b+7, "%02X", type);
  int i = 0;
  while (i < len*2){
    sprintf(b+9+i, "%c", cmd[i]);
    i++;
  }
  sprintf(b+9+i, "%02X\n", (unsigned char)checksum(len, b));
  fprintf(fout,b);
}
```

Figura 3.2: Código de la función write_record

```
int checksum(int len,char b[256]){
  int sum = 0, partial;
  int i=0;
  char c;
  //b+1 saltea el ':'
  while (i < (len+4)*2){
    sscanf((b+1+i), "%02X", &partial);
    sum += partial;
    i+=2;
  }
  return -sum;
}
```

Figura 3.3: La función de cálculo del checksum

3.4. *udev* y *fxload* para la carga del firmware

Otros modelos de cámaras del mismo fabricante soportadas en GNU/Linux tienen solucionado el problema de detectar la conexión del dispositivo a algún puerto USB y la carga del firmware para dicho dispositivo mediante la utilización del mecanismo *udev* de GNU/Linux. Existía por lo tanto un archivo de reglas para *udev* el cual fue modificado agregando las acciones necesarias para cargar el firmware para la QHY5T en forma automática al momento de la conexión al puerto USB.

Dicha modificación consiste en agregar dos líneas al archivo **85-qhy.rules** el cual se adjunta con este trabajo. Una de dichas líneas es:

```
ATTRS{idVendor}=="1618", ATTRS{idProduct}=="0910", RUN+="/sbin/fxload -t fx2 -I\
                        /etc/qhyccd/qhy5t.hex -D $env{DEVNAME}"
```

Esta especifica que al momento en que se detecte la conexión al puerto USB de un dispositivo con VID:PID[3] = 1618:0910, se debe correr el comando detallado en el parámetro RUN, el cual carga el firmware al dispositivo mediante una llamada a *fxload*. El VID:PID se averiguó mediante la utilización de la herramienta *lsusb*. La carga del firmware no modifica estos valores.

La otra línea agregada es:

```
ATTRS{idVendor}=="1618", ATTRS{idProduct}=="0910*", OWNER="root", GROUP="video",\
MODE="0664", SYMLINK+="QHY5T"
```

[3]Vendor ID: Product ID

Esta configuración permite que el dispositivo sea usado por los usuarios del grupo `video` y no exclusivamente por el usuario `root`. Posteriormente esta configuración se hizo estándar para todos los demás dispositivos soportados por el fabricante y es un aporte más de este trabajo a la comunidad.

3.5. Publicación del firmware del fabricante

Pasado un año del comienzo de las investigaciones para extraer el firmware mediante las técnicas antes mencionadas, el fabricante cedió ante los pedidos de la comunidad de usuarios con sistemas GNU/Linux e hizo disponible una versión oficial del firmware en formato IHEX. Dado que las funciones soportadas por la cámara dependen en gran medida del firmware y dado que no se liberó el código fuente no pueden agregarse funciones nuevas, por ejemplo soporte para binning 3x3 o desplazamientos de canales, funciones que si bien están soportadas por el sensor, quedan ocultas por la abstracción del firmware.

Para entonces ya se había empezado a hacer ingeniería inversa del firmware mediante el proceso de de-compilación. Sin embargo el análisis de las funciones del firmware no revelaba información relevante acerca de los cálculos de tiempo en relación al tiempo de exposición y el tamaño del frame. Se había logrado obtener frames válidos desde el sensor duplicando la información de configuración capturada mediante análisis de tráfico y si bien era posible modificar la ganancia arbitrariamente, cambios muy pequeños a cualquier otro parámetro producía frames inválidos. La figura 3.4 muestra un ejemplo de los datos que podían ser deducidos por medio del análisis de tráfico USB.

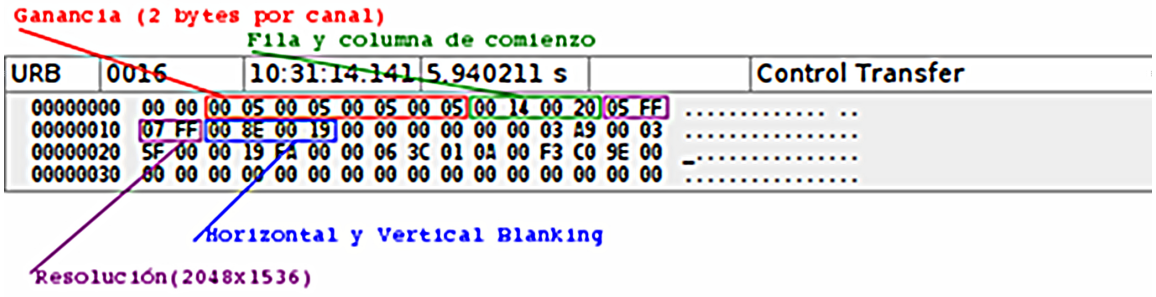


Figura 3.4: Identificación de los parámetros de configuración enviados a la cámara mediante análisis de tráfico USB.

Hasta el momento en que el fabricante hizo disponible el código fuente del driver para Windows y durante dos meses, todo avance respecto al controlador para GNU/Linux estuvo parado y si bien el fabricante respondía las consultas, lo hacía en tiempos dilatados y las respuestas carecían de detalles suficientes para salir adelante.

Además del firmware, el fabricante hizo disponible para este trabajo el código fuente de los controladores para Windows, con la condición que no fueran hechos públicos.

Capítulo 4

El driver

El controlador desarrollado para este proyecto esta basado en primer lugar en el código del controlador para una cámara de la misma familia, la QHY5. El controlador para esta cámara fue escrito en 2009 por Geoffrey Hausheer y publicado con una licencia GPL2. Este driver cuenta con algunas funciones interesantes, por ejemplo la relacionada con manejadores (handles) USB que permiten buscar dispositivos conectados, abrir y cerrar manejadores para un dispositivo en particular, leer y escribir datos en conexiones USB establecidas y guardar archivos a disco en formato .pgm, entre otras. Además se incluye un main rudimentario a modo de pruebas que permitía testear todas las funciones del driver. El estudio de este controlador fue el primer contacto del autor con la librería libusb.

El presente trabajo también está basado en el controlador para Windows para la cámara QHY5T, el cual fue desarrollado entre 2007 y 2009 por Tom Van den Eede para la empresa QHYCCD. Dicho controlador fue cedido para este trabajo por el fabricante, con la condición que dicho código no se haga público, debido a la responsabilidad adquirida por dicha empresa con el fabricante Orion, con el cual tenía acuerdos en relación al modelo de la cámara en cuestión.

El hecho de que el fabricante hiciera disponible el código fuente del driver manifiesta el interés por su parte de dar soporte al sistema operativo GNU/Linux. Además gracias a esto y a las consultas realizadas a Tom por correo electrónico y chat es que fue posible tener un driver para GNU/Linux en pocos meses y aceleró el proceso de desarrollo respecto a las técnicas de ingeniería reversa las cuales habían dado buenos resultados pero en un tiempo considerablemente mayor.

Se detallan a continuación las funciones más importantes del driver. En muchos casos se hablarán de cambios realizados en relación a las primeras versiones, cuando esto sea relevante.

Cabe resaltar que si bien el driver permite controlar las principales funciones de la cámara, el autor considera este trabajo como "trabajo en progreso". Algunas de las características soportadas en el controlador para Windows no están contempladas en el driver para GNU/Linux, a saber el soporte para imágenes de 16 bits y el soporte de binning 2x2. Además, un proceso de refactorización permitiría incluir soporte para múltiples dispositivos conectados a la misma computadora y compatibilizar la API con la del driver para GNU/Linux de la cámara QHY5, además de las características antes mencionadas.

4.1. Detalles de la implementación

El driver para GNU/Linux para la cámara QHY5T está escrito en lenguaje C89, al igual que el driver para la cámara QHY5. A diferencia de este último está dividido en dos archivos, `qhy5t.c` y `qhy5t.h`, separando así la implementación del driver de la API para desarrollo. Una aplicación que use el driver importa el archivo `qhy5t.h` el cual contiene los encabezados de todas las funciones y los tipos exportados por el driver, mientras en el archivo `qhy5t.c`, además de implementar estas funciones, implementa otras que sirven de base al funcionamiento del controlador (como las funciones de comunicación USB o la sincronización de threads) pero que permanecen ocultos a las aplicaciones usuarias del mismo.

4.2. El tipo qhy5t_driver

El tipo `qhy5t_driver` es un sinónimo de la estructura anónima más importante del driver. Representa el estado interno de los registros de la cámara. La figura 4.1 muestra la declaración de la estructura en el archivo `qhy5t.h`.

```
typedef struct {
  usb_dev_handle *handle;
  uint16_t width;
  uint16_t height;
  uint16_t offw;
  uint16_t offh;
  uint8_t binmode;
  uint16_t gg1;
  uint16_t gb;
  uint16_t gr;
  uint16_t gg2;
  uint16_t vblank;
  uint16_t hblank;
  uint8_t bpp;
  uint16_t etime;
  void *image;
  size_t framesize;
}qhy5t_driver;
```

Figura 4.1: Declaración de la estructura qhyt5_driver

En esta estructura, el puntero `handle` es el punto de comunicación con el dispositivo USB asociado con la cámara. Su valor se asigna si la función `qhy5t_open()` tiene éxito. Todas las funciones de transferencia de datos entre la computadora y el dispositivo se realizan mediante la librería libusb, las cuales utilizan un puntero a `usb_dev_handle`. Este último es uno de los tipos principales de dicha librería.

Los miembros `width`, `height`, `offw` y `offh` determinan el tamaño y la posición del frame. Los rangos para `width` y `height` son (4, 2048) y (4, 1536) respectivamente. Los rangos para `offw` y `offh` son entre (0, 2048-width) y (0, 1536-height).

El campo `binmode` indica el binning. Por defecto es 1, representa binning 1x1 y es el único modo soportado hasta el momento de escribir este trabajo.

Los miembros `gg1`, `gb`, `gr` y `gg2` representan la ganancia de los diferentes colores de pixels del sensor. Dado que este valor no es lineal, se debe configurar invocando a la función `set_gain()`, la cual es detallada más adelante. El rango de valores para la ganancia es de (1, 167) y se puede configurar independientemente para cada color.

`vblank` y `hblank` representan los tiempos para calcular el vertical blank y el horizontal blank en la función `program_camera()`. Por defecto, `vblank` toma el valor 25 y `hblank`, el valor 0. Estos valores son los mismos utilizados en el controlador para Windows y este trabajo no provee una forma de modificar estos valores garantizando la integridad de los frames.

El campo `bpp` hace referencia a los bits por pixel. Por defecto es 8, indicando que cada pixel de la imagen es de 1 byte. Es el único modo soportado hasta el momento de escribir este trabajo.

El miembro `etime` es el que representa el tiempo de exposición en milisegundos. El rango es de 1 a 60000, es decir que se permiten tomas de 1 milisegundo a 60 segundos de exposición. El problema de hacer exposiciones de más de varios segundos es que el ruido tiende a igualar y en algunos casos a superar a la señal capturada.

Por último, el miembro `image` es un puntero a un buffer de tamaño `framesize` bytes.

Casi todas las funciones implementadas para el controlador reciben un puntero a `qhy5t_driver`, con excepción de `qhy5t_open()`, la cual devuelve un puntero de este tipo y de `write_pgm()`.

4.3. Las funciones exportadas

El controlador presentado en este trabajo fue diseñado para ser utilizado por una aplicación externa, la cual a través de la inclusión de los encabezados de las funciones tendrá disponibles todas las funciones necesarias para acceder al dispositivo. Las funciones exportadas por el driver son aquellas que otro programa puede utilizar con solo incluir el archivo `qhy5t.h` mediante una directiva `include`.

De las más de quince funciones implementadas en el driver, solo diez son exportadas en el archivo de encabezados. Estas se describen brevemente a continuación, en el orden normal de invocación.

- `qhy5t_open()`: Esta función no recibe argumentos y devuelve un puntero a `qhy5t_driver`, que es NULL si no hay un dispositivo QHY5T conectado al bus USB de la computadora. Se debe invocar antes que las demás funciones.

- `qhy5t_set_params()`: Esta función recibe los parámetros de configuración para la cámara y los guarda en la estructura `qhy5t_driver`.

- `qhy5t_program_camera()`: Esta función escribe efectivamente los parámetros para las capturas en los registros de la cámara, además de realizar algunos cálculos relacionados con los tiempos del sensor. Esta función se invoca una vez, luego de invocar a `qhy5t_set_params()`.

- `qhy5t_start_capture()`: Esta función se encarga de iniciar el thread de exposición, una vez que la cámara a sido programada con los parámetros de captura deseados. El thread estará activo hasta que se invoque a la función `qhy5t_stop_capture()`, adquiriendo frames con dichos parámetros.

- `qhy5t_read_exposure()`: Esta función se invoca cada vez que el usuario del driver requiera un frame. Se comunica a través de semáforos con el thread de exposición para acceder de manera exclusiva al buffer desde el cual son copiados los datos.

- `write_pgm()`: Esta función escribe los datos de la imagen en un archivo. Para esto recibe como parámetros el buffer que contiene la imagen y sus dimensiones, además de el nombre del archivo a crear.

- `qhy5_timed_move()`: Esta función envía comandos de guiado al puerto ST-4. Recibe como parámetros las direcciones del movimiento y su duración en milisegundos.

- `qhy5_cancel_move()`: Esta función cancela cualquier comando de guiado.

- `qhy5t_stop_capture()`: Esta función detiene el thread de captura.

- `qhy5t_close()`: Esta función cierra la conexión con el dispositivo y libera la memoria alocada para las estructuras.

A continuación se detallan las funciones implementadas, tanto las que pertenecen a la API de programación como las funciones internas de comunicación y sincronización del controlador.

4.4. La función ctrl_msg()

A través del uso de esta función se realizan todos los envíos y recepciones de mensajes de control entre el dispositivo y la computadora. Es un *wrapper* de la función `usb_control_msg()` de la librería libusb y que incluye además información de depuración para los mensajes de control. La versión de libusb utilizada es la 0.1 la cual está en desuso y está siendo reemplazada por libusb 1.0. La migración del driver a la nueva versión de la librería libusb está planeada para el futuro cercano.

La implementación de la función se muestra en la figura 4.2.

```
int ctrl_msg(usb_dev_handle *handle, unsigned char request_type,
             unsigned char request, unsigned int value,
             unsigned int index, uint8_t *data, unsigned char len){
  int i, result;
  static int countmsg=0;
  dprintf("\n");
  dprintf("Sending %s command 0x%02x, 0x%02x, 0x%04x, 0x%04x,
           %d bytes\n",
           (request_type & USB_ENDPOINT_IN) ? "recv" : "send",
            request_type, request, value, index, len);
  result = usb_control_msg(handle, request_type,
           request, value, index, (char*)data, len, 5000);
  for(i = 0; i < len; i++) {
    dprintf(" %02x", (unsigned char)data[i]);
  }
  dprintf("\n");
  dprintf("msj# %d: Bytes writen = %d\n", countmsg, result);
  countmsg++;
  return result;
}
```

Figura 4.2: Implementación de la función `ctrl_msg()`

La información de depuración incluida en esta función fue de suma importancia durante las primeras etapas del desarrollo y durante el proceso de ingeniería inversa, donde era posible comparar la información enviada y recibida respecto a las capturas de tráfico obtenidas usando USBlyzer. El modelo de esta función fue tomado del driver para la cámara QHY5, la cual fue modificada para incluir dicha salida de depuración.

La macro dprintf expande a printf si debug es mayor que cero. Está definida de la siguiente manera:

```
#define dprintf if(debug > 0) printf
```

donde debug es una variable global que se inicializa a 1 si el programa se compila con la opción `-DDEBUG` y a cero en caso contrario.

Durante la etapa de estudio del driver para Windows, se detectaron varias funciones para realizar comunicaciones con el dispositivo a través del puerto USB. Estas comunicaciones pueden dividirse en dos grandes grupos, aquellas donde se transmiten datos en modo *BULK*[1], ya sea hacia o desde el dispositivo y aquellas en las que se envían o reciben mensajes de control.

En este trabajo se utilizaron directamente las funciones `usb_bulk_write` y `usb_bulk_read` provistas por la librería libusb, para cubrir las necesidades del primer tipo de tráfico. Esto además es consistente con la implementación del driver para GNU/Linux de la cámara QHY5.

Para el segundo tipo de tráfico se utilizó la función `ctrl_msg()` la cual recibe como parámetro un puntero a `usb_dev_handle` cuyo nombre es `handle`, un tipo de requerimiento (generalmente, `read` o `write`), un número que indica el requerimiento, el valor de mensaje de control, el índice, un buffer para enviar o recibir y la longitud.

Una cosa que parecía sencilla pero que resultó bastante molesta fue que en el controlador de referencia para el sistema operativo Windows se utilizaba comúnmente una función que realizaba la misma tarea pero cuyos argumentos estaban en distinto orden. Mientras que en dicha función los argumentos son `type, request, buffer, size, index` y `value`, en la función `ctrl_msg()` dichos argumentos son `type, request, value, index, buffer` y `size`. Llevó mucho tiempo y esfuerzo traducir los llamados y fueron necesarias varias horas de revisión de código, ya que `type, request, value, index` y `size` son números y muchas veces aparecen hardcodeados en la versión del driver para Windows.

[1]El modo *BULK* se utiliza en transferencias de datos grandes (de hasta 512 bytes), en ráfagas . Está garantizada la detección de errores y la retransmisión pero no el ancho de banda o la mínima latencia.

4.5. open y close

El driver exporta dos funciones para abrir y cerrar la conexión al dispositivo. La función `qhy5t_open()` devuelve un puntero a un `qhy5t_driver` si la apertura tiene éxito o `NULL` en caso contrario. Esta función hace un llamado a `qhy5t_locate()`(ver Figura 4.4) la cual se encarga de determinar si existe alguna cámara QHY5T conectada y devuelve un puntero a un `usb_dev_handle` si tiene éxito. Este puntero es asignado al miembro `handle` de la estructura `qhy5t_driver`, luego de reservar espacio en memoria para dicha estructura. Un fragmento de la función `qhy5t_locate()` sin las directivas de depuración se muestra en la figura 4.3. La lógica de esta función fue tomada de la función `locate_device()` del driver para GNU/Linux de la cámara QHY5.

La función `qhy5t_close()` simplemente invoca a `usb_close(qhy5t->handle)`, la cual es la forma tradicional de indicar que el `usb_dev_handle` ya no será utilizado. Además, `qhy5t_close()` desaloca la memoria reservada por la llamada a `qhy5t_open()`.

```
usb_dev_handle *qhy5t_locate(unsigned int vid, unsigned int pid){
  unsigned char located = 0;
  struct usb_bus *bus;
  struct usb_device *dev;
  usb_dev_handle *device_handle = NULL;

  usb_find_busses();
  usb_find_devices();
  for (bus = usb_busses; bus; bus = bus->next)
  {
    for (dev = bus->devices; dev; dev = dev->next){
      if (dev->descriptor.idVendor == vid && dev->descriptor.idProduct
          == pid){
        located++;
        device_handle = usb_open(dev);
      }
    }
  }
  if (device_handle==NULL){
    return NULL;
  }
  usb_set_configuration(device_handle, 1);
  usb_claim_interface(device_handle, 0);
  usb_set_altinterface(device_handle, 0);
  return device_handle;
}
```

Figura 4.3: Esta función es invocada desde `qhy5t_open` de esta manera mostrada en 4.4

```
if ((handle = qhy5t_locate(0x1618, 0x0910))==NULL){
  printf("Could not open the QHY5T device\n");
  return NULL;
}
```

Figura 4.4: Invocación a locate desde `qhy5t_open`

La función `qhy5t_open` debe ser invocada en primer lugar ya que todas las demás funciones utilizan de alguna manera el puntero devuelto por ésta. Por otra parte, la función `qhy5t_close` es la responsable de cerrar las conexiones y liberar los recursos reservados por las funciones anteriores. La figura 4.5 muestra la implementación de dicha función.

```
void qhy5t_close(qhy5t_driver *qhy5t){
  if(! qhy5t)
    return;
  qhy5t_reconnect(qhy5t);
  if(qhy5t->handle)
    usb_close(qhy5t->handle);
  free(qhy5t);
  return;
}
```

Figura 4.5: Implementación de la función `qhy5t_close`

4.6. set_params() y program_camera()

Existen dos funciones cuyo uso permite programar la cámara con determinados parámetros como tiempo de exposición, tamaño del cuadro, ganancia del sensor, etc. La función `qhy5t_set_params()` es la encargada de llenar los datos de la estructura `qhy5t_driver` con los datos que recibe como parámetro. Lo hace después de controlar que dichos parámetros estén dentro de los límites y si no es así, inicializa los campos con valores seguros.

La implementación de `qhy5t_set_params()` puede verse en la figura 4.6

Notar que para todos los datos se utilizan tipos `uint_8` o `uint_16`. Estos tipos están definidos en stdint.h[2] y está garantizado que el tamaño de ambos es de 1 y 2 bytes respectivamente.

La función anterior sin embargo, no escribe estos datos en los registros. Para esto, es necesario invocar a la función `qhy5t_program_camera()`. Esta función solo recibe dos parámetros, un puntero a `qhy5t_driver` y un entero que indica si la cámara debe reprogramarse. Dados los valores contenidos en la estructura `qhy5t_driver`, la función genera un buffer de datos el cual luego es transferido a la cámara utilizando la función `ctrl_msg()`. Esta función fue traducida casi literalmente desde el driver de Windows ya que realiza algunos cálculos que no están documentados más que en el código fuente mismo[3].

Algunos cálculos realizados por esta función son sencillos. Por ejemplo, si se ha configurado la cámara para tomar frames de 2048x1536 pixels, el miembro width de la estructura qhy5t_driver tendrá el valor 2048. Sin embargo, el valor a escribir en el registro (Reg0x04) de configuración del sensor debe ser 2047, es decir que se debe restar 1 al valor contenido en el miembro de la estructura. Otro ejemplo son los registros de desplazamiento. Dado el desplazamiento en x e y, se debe sumar 0x20 y 0x14 respectivamente, para evitar leer los pixels de calibración.

Otros cálculos son mucho más oscuros. Por ejemplo, los relacionados con los tiempos de validez del frame o el tiempo de reset (antes del cual el frame debe ser leído o se pierde) o el tamaño efectivo de la lectura, el cual en general es más grande que `width` x `height`.

Estos cálculos fueron incorporados a la función casi directamente. Solo fue necesario reemplazar algunos tipos ya que en el controlador para Windows se utilizaba por ejemplo DWORD para hacer referencia a datos de 16 bits. Además en dicho controlador se encuentran separados por una sentencia switch/case los cálculos de tiempos de pre y post blanking, reset y tamaño del frame, además de algunas configuraciones para los registros GPIF del procesador de la cámara, dependiendo de la configuración del binning, 1x1 o 2x2. Por comodidad y pensando en la futura implementación de binning 2x2, la sentencia switch/case se implementó en este trabajo, aunque solo tiene una rama con los cálculos correspondientes para binning 1x1.

Una vez realizados todos los cálculos se genera un vector de 64 bytes donde se cargan los datos calculados en un orden preestablecido en formato big endian. Para la conversión de little endian a big endian se utilizaron las mismas macros disponibles en el controlador para Windows, las cuales están definidas como se muestra en la figura 4.7. Notar que estas macros hacen referencia explicita a las variables `bi` y `buffer`, las cuales deben estar declaradas en la función que las utilice.

[2] El encabezado de stdint.h es incluido en usb.h.

[3] Muchos de los valores involucrados en dichos cálculos, incluso están hardcodeados sin que medie comentario alguno.

```
void qhy5t_set_params( qhy5t_driver *qhy5t,
               uint16_t w,
               uint16_t h,
               uint16_t x,
               uint16_t y,
               uint8_t bin,
               uint16_t gg1,
               uint16_t gb,
               uint16_t gr,
               uint16_t gg2,
               uint16_t vblank,
               uint16_t hblank,
               uint8_t bpp,
               uint16_t etime){

    w = qhy5t->width = (w >= 4 && w <=2048) ? w : 2048;
    h = qhy5t->height = (h >= 4 && h <=1536) ? h : 1536;
    qhy5t->offw =  (x >= 0 && x <=(2048 - w)) ? x : (2048 - w) / 2;
    qhy5t->offh =  (y >= 0 && y <=(1536 - h)) ? y : (1536 - h) / 2;
    qhy5t->binmode = 1;
    qhy5t->gg1 = (gg1 > 0 && gg1 < 0x7860) ? gg1 : 1;
    qhy5t->gb = (gb > 0 && gb < 0x7860) ? gb : 1;
    qhy5t->gr = (gr > 0 && gr < 0x7860) ? gr : 1;
    qhy5t->gg2 = (gg2 > 0 && gg2 < 0x7860) ? gg2 : 1;
    qhy5t->vblank = vblank;
    qhy5t->hblank = hblank;
    qhy5t->bpp = 8;
    if (etime > 1)
        qhy5t->etime = etime;
    if (etime > 60000)
        qhy5t->etime = 60000;
    qhy5t->image = calloc(2,w*h);
    qhy5t->framesize=0;
}
```

Figura 4.6: Implementación de la función `qhy5t_set_params`. Esta función controla que los parámetros configurados sean seguros.

```
#define HHSB(x) ((uint8_t)(((uint32_t)x>>24)&0xFF))
#define HSB(x) ((uint8_t)(((uint32_t)x>>16)&0xFF))
#define MSB(x) ((uint8_t)(((uint32_t)x>>8)&0xFF))
#define LSB(x) ((uint8_t)(((uint32_t)x)&0xFF))

#define BUFBYTE(W) buffer[bi++] = LSB(W)
#define BUFWORD(W) buffer[bi++] = MSB(W); buffer[bi++] = LSB(W)
#define BUFTRIP(W) buffer[bi++] = LSB(W); buffer[bi++] = MSB(W);
   buffer[bi++] = HSB(W)
#define BUFWORDR(W) *((uint16_t *)&buffer[bi]) =W ; bi+=2;
#define BUFDWORD(W) buffer[bi++] = HHSB(W); buffer[bi++] = HSB(W);
   buffer[bi++] = MSB(W) ;buffer[bi++] = LSB(W)
```

Figura 4.7: Macros utilizadas para llenar el vector de 64 bytes con los valores de configuración en formato big endian.

La función además recibe un parámetro que permite reprogramar la cámara con los últimos valores utilizados. Esto es porque el buffer con los valores calculados es copiado a un arreglo estático interno de la rutina `qhy5t_program_camera()`. Además este arreglo permite programar la cámara solamente si los valores calculados han cambiado.

Si bien la función adolece de varios vicios de programación, carece de estilo y necesita una seria refactorización, como caja negra cumple su tarea diligentemente. La función puede verse en los archivos de código entregados.

4.7. La función main() del driver

El driver para la QHY5 incluye una implementación de la función `main()`. Esto permite depurar el driver compilando con la opción `-DQHY5_TEST`, a la vez que muestra una implementación de referencia para futuras aplicaciones clientes del driver.

El programa así generado es utilizado por la aplicación *Qhyimager*. Este programa escrito por Dan Höller y Clive Rogers entre otros, utilizando la suite de programación Real Basic, es una interfaz gráfica para captura de larga exposición que soporta algunos modelos de cámaras de la marca QHYCCD. Dado que los drivers de otras cámaras también incluyen una función `main()`, *Qhyimager* utiliza llamadas al sistema para invocar a estos subprogramas de captura utilizando como parámetro (a `main()`) la opción `-c 1`. Este parámetro es soportado por la función `main()` de todos los drivers y permite que se grabe un frame a disco con el nombre de archivo seleccionado. Debido al sistema de llamadas de *Qhyimager*, por cada frame es necesario abrir la conexión a la cámara, programarla, obtener el frame y cerrar la conexión, lo que hace que la QHY5T (o cualquier otra cámara de las soportadas) funcione a un frame rate muy por debajo del rendimiento de su contraparte para Windows, incluso en resoluciones bajas. Esto, que no es problema en fotografías de larga exposición donde se requiere un frame cada varios segundos, no es adecuado para fotografía lunar, solar o planetaria.

El controlador para este trabajo incluye una versión modificada para esta función, lo que permite probar las funciones implementadas de manera práctica y es posible que el programa devuelva más de un frame y los guarde en archivos antes de cerrar la conexión (por ejemplo, invocando al programa con -c y el número de frames deseados). Sin embargo, no es posible ver lo que se fotografía hasta después que el programa termine y sólo a través de un visor de archivos pgm o FITS externo. Por este motivo se desarrolló el programa qhy5tviewer por separado. Este programa se describe en el capítulo 5 y permite además de ver en tiempo real los archivos que se van guardando, acceder a otras funciones del driver como los comandos de guiado.

4.8. El proceso de adquisición

Una mejora respecto al controlador para la cámara QHY5 y otros similares para GNU/Linux es la utilización de un thread separado dedicado para la captura de frames. Esta idea fue tomada del driver para Windows para la QHY5T pero en lugar de usar la API `process.h` se utilizó la librería de threads estándar *POSIX* `pthread.h`.

La librería `pthread.h` provee mecanismos para crear hilos de ejecución separados, los cuales luego son mapeados a threads físicos por el sistema operativo que corren en procesadores separados si es posible. Además, esta librería provee también mecanismos para sincronizar y comunicar estos hilos de ejecución, como semáforos, variables condición y barreras.

La solución pensada para este trabajo consiste en separar el thread de exposición del resto de la lógica del driver. Esto se logra creando un thread para la exposición, el cual es instanciado llamando a la función `qhy5t_start_capture()` luego de que la cámara es configurada.

Este es un ejemplo clásico del problema del productor/consumidor en el paradigma de programación concurrente. El thread de exposición hace las veces de productor mientras que la función `qhy5t_read_exposure()` cumple el papel del consumidor.

Hay tres buffers en juego. El primero es un buffer local a la función `qhy5t_exposure_thread()` y es sobre el cual se leen los datos de la cámara invocando la función USB_BULK_READ. Dado que este buffer es local, no se encuentra dentro de la sección crítica. Una vez que se comprueba que la lectura fue exitosa, se copian los datos de este buffer a un buffer global. Dado que el buffer global es accedido también por la función `qhy5t_read_exposure()`, éste si se encuentra dentro de la sección crítica y por

ello, ambas funciones deben lograr el acceso exclusivo a dicha sección. Para esto y como es la manera más sencilla de hacerlo en pthreads, se utilizan semáforos (*mutex*).

Los *mutex* en pthreads tienen dos funciones principales, `pthread_mutex_lock()` la cual detiene la ejecución del thread hasta que pueda bloquear el semáforo[4] y `pthreads_mutex_unlock()` la cual libera el semáforo. De esta manera se puede lograr exclusión mutua en una sección crítica utilizando dos variables *mutex* que en el driver reciben el nombre de `reading` y `writing`.

El thread de exposición es creado cuando se invoca a la función `qhy5t_start_capture()`, la que además aloca memoria para el buffer global e invoca a `pthread_mutex_lock()` sobre el semáforo `reading`. Esto garantiza que de las funciones `qhy5t_exposure_thread()` y `qhy5t_read_exposure()`, es la primera la que logra el acceso antes a la sección crítica. Esto tiene sentido ya que de no ser así, se corre el riesgo de tratar de leer un frame que todavía no fue capturado.

La función `qhy5t_exposure_thread()` toma el control de la sección crítica invocando a la función `pthread_mutex_lock()` sobre el semáforo `writing`. Luego copia el buffer local que contiene la información de la última exposición en el buffer global. Esta copia no es directa si no que requiere que la información del buffer leído con USB_BULK_READ() se mapee a un frame de las dimensiones correctas descartando el *overscan*. Por último, el thread de exposición devuelve el control de la sección crítica invocando a `pthreads_mutex_unlock()` sobre el semáforo `reading`.

Como consecuencia, la función `qhy5t_read_exposure()` ahora puede acceder a la sección crítica invocando a `pthread_mutex_lock()` sobre el semáforo `reading`. Una vez conseguido el acceso, la función copia la información del buffer global al miembro de la estructura del driver `qhy5t->image` y libera el acceso a la sección crítica invocando a `pthread_mutex_unlock()` sobre el semáforo `writing`.

La estructura del proceso de captura en pseudocódigo es la siguiente

```
start_capture(){
  lock(reading);
  //creación del thread de exposición (llamada a exposure_thread())
}

read_exposure(qhy5t){
  lock(reading);
  //comprobación de los datos
  //copia del buffer global a qhy5t->image
  unlock(writing);
  }
}

exposure_thread(){
  while (true){
    //adquisición de los datos del sensor al buffer local
    lock(writing);
    //comprobación de los datos
    //mapeo del frame del buffer local al global
    unlock(reading);
  }
}

stop_capture(){
  //señal de fin de la captura
  //liberación de recursos
}
```

La última función relacionada con el proceso de adquisición de datos es `qhy5t_stop_capture()` la cual detiene la captura en forma limpia, es decir, cancela el thread de exposición.

[4]Solo un hilo a la vez puede bloquear un semáforo.

4.9. La función set_gain()

La ganancia del sensor tiene un rango de 1 a 128, pero este rango no es lineal. La ganancia de 1 a 4 puede incrementarse en pasos de 0.125, de 4.25 a 8 en pasos de 0.25 y de 9 a 128 en pasos de 1. Esto da un total de 167 pasos. Para simular un rango lineal utilizando los valores recomendados en la tabla 2.2 se desarrolló la función `qhy5t_set_gain()`.

La función toma un valor entero y devuelve el valor recomendado para el paso correspondiente. La figura 4.8 muestra la implementación de dicha función.

```
int qhy5t_set_gain(int gain){
//de los valores recomendados del datasheet del sensor
  if (gain <= 0){
    return 0x0008;
  }
  if (gain <= 24){ // gain 1 - 4, step 0.125
    return 0x0008 + gain;
  }
  if (gain <= 24 + 15){ //gain 4.25 - 8, step 0.25
    return 0x0051 + (gain - 24);
  }
  if (gain <= 24 + 15 + 128){ // gain 9 - 128, step 1
    return 0x60 + ((gain - 24 - 15) << 7);
  }
  if (gain > 24 + 15 + 128){
    return 0x7860;
  }
}
```

Figura 4.8: La función `qhy5t_set_gain()`

Esta función es muy diferente a las encontradas en los drivers de referencia. En el driver para Windows para la QHY5T el valor de la ganancia es pasado al registro de la cámara como viene, con lo que probablemente sea responsabilidad del programa de captura elegir los valores apropiados para la configuración. Tampoco se especifica el rango de la ganancia pero por consultas con el desarrollador se había determinado que un rango de 0 a 1000 era apropiado.

En la versión para GNU/Linux para la QHY5, la ganancia se configura utilizando el parámetro como un índice de desplazamiento en un arreglo que contiene todos los posibles valores de configuración de la ganancia para el sensor de dicha cámara, que recordemos, es diferente de la QHY5T dado que ambas cámaras utilizan sensores de modelos diferentes.

Las primeras versiones de la función `qhy5t_set_gain()` utilizaron ambas aproximaciones a la solución. Sin embargo los resultados obtenidos fueron no satisfactorios, ya que el control de la ganancia carecía de linealidad y por ejemplo, valores por debajo de 500 producían frames con más ganancia que los valores por arriba de 500. Solo recientemente y a través del estudio detallado del datasheet fue posible escribir una función que no solo devolviera valores lineales para la ganancia en pasos enteros, sino que fueran óptimos en cuanto a la relación señal ruido.

4.10. La función write_pgm()

El controlador de referencia de la cámara QHY5 incluye una función para guardar los frames a disco utilizando el formato pgm del conjunto de formatos *Netpbm*[9]. Esta función se denomina erróneamente `write_ppm()`, el cual es un formato color, mientras escribe un archivo pgm, el cual es un formato en escala de grises. Dado que la cámara QHY5 tiene un sensor monocromático, esta función cumple perfectamente su objetivo de guardar los datos del sensor en un archivo a pesar de su nombre. Para el driver presentado en este trabajo, se renombró la función a `write_pgm()`.

El conjunto de formatos Netpbm incluye los formatos pbm, pgm, ppm y recientemente, pam. Los tres primeros son muy similares y muy sencillos en cuanto a su especificación, la cual puede encontrarse en [10], [11] y [12] respectivamente. Por su importancia para este trabajo, sin embargo, se describe brevemente el formato pgm, utilizado por esta función.

El formato pgm

Pgm es un formato para representación de imágenes en escala de grises sumamente sencillo. Tiene dos especificaciones, una binaria (o RAW) y la otra en formato ASCII. Una imagen pgm esta formada por:

- Dos bytes para el magic number que indican que se trata de un archivo pgm. Para el formato binario se corresponde con la cadena de caracteres "P5".
- Un espacio en blanco (se aceptan como espacios en blanco uno o varios espacios, tabulaciones o caracteres de fin de linea "CR" o "LF").
- El ancho (*width*) de la imagen en formato decimal codificado en ASCII.
- Un espacio en blanco.
- El alto (*height*) de la imagen en formato decimal codificado en ASCII.
- Un espacio en blanco.
- Un número en formato decimal codificado en ASCII indicando el mayor valor posible para el nivel de gris de la imagen. Este número se conoce como *maxval* y debe ser menor que 65536 y mayor que 0.
- Un único carácter en blanco, generalmente un carácter de fin de linea.
- Una matriz (*raster*) de *height* filas, en orden de arriba hacia abajo. Cada fila consiste de *width* valores de gris, en orden de izquierda a derecha. Cada valor de gris representa un pixel cuadrado de la imagen, en la cual los pixels están contiguos, y es un número entre 0 y *maxval* con 0 representando el color negro y *maxval* el valor blanco. Cada valor de gris se representa en formato binario, en un byte si *maxval* es menor que 256 y en dos bytes en caso contrario, con el byte más significativo primero.

Una variación de este formato es el pgm plano, el cual tiene un magic number "P2" y en el el valor de gris para cada pixel es un número decimal entre 0 y *maxval* representado en formato ASCII. Cada pixel a su vez está separado por un espacio en blanco. Esto permite que una imagen en este formato pueda modificarse utilizando un simple editor de texto. Sin embargo una imagen en este formato utiliza al menos el doble de espacio que la misma imagen en formato pgm binario.

Modificaciones a la función original

Uno de los cambios es respecto al nombre de la función, la cual pasó a llamarse `write_pgm()`. Tanto la firma como el cuerpo de la función fueron modificados levemente para que sea más compacta y eficiente, pero la salida de la versión original es exactamente la misma respecto a la versión modificada.

La figura 4.9 muestra un fragmento de la versión modificada mientras que la figura 4.10 muestra la versión original. Cabe destacar los cambios realizados a la firma de la función donde se reemplaza el primer parámetro por un puntero a void que representa los datos a escribir en la imagen. Este cambio es porque como se ve en la sección 4.8, los datos correspondientes a un frame a escribir no necesariamente están en el campo `image` de la estructura que representa al dispositivo.

```
void write_pgm(void * data, int width, int height, char *filename){
  FILE *h = fopen(filename, "w");
  fprintf(h, "P5\n"); //ppm header
  fprintf(h, "%d %d\n", width, height);//ppm header
  fprintf(h, "255\n");//ppm header
  fwrite(data, width*height, 1, h);
  fclose(h);
}
```

Figura 4.9: La función `write_pgm()`.

```
void write_ppm(qhy5_driver *qhy5, int width, int height, char *
    filename){
  int row, col;
  FILE *h = fopen(filename, "w");
  fprintf(h, "P5\n");
  fprintf(h, "%d %d\n", width, height);
  fprintf(h, "255\n");
  for(row = 0; row < height; row++) {
    unsigned char *ptr = qhy5_get_row(qhy5, row);
    for(col = 0; col < width; col++)
      fprintf(h, "%c", *(ptr++));
  }
  fclose(h);
}
```

Figura 4.10: La función `write_ppm()` del controlador de referencia.

Notar el uso de la función `qhy5_get_row()` en la función original de la figura 4.10. En este controlador, es esta función la que se encarga de mapear el frame eliminando los datos de *overscan*. Dado que en el presente trabajo, esta operación se realiza directamente en el thread de exposición, la función simplemente puede escribir todo el buffer a la vez sin necesidad de recurrir a un `for` para escribirlo fila por fila.

Una modificación interesante a realizar en el futuro es hacer el chequeo de que el archivo se pudo abrir con éxito, para evitar una posible violación de segmento si la apertura falla. Esta modificación parece muy sencilla pero podemos ver fácilmente que no lo es. La opción de poner un `if` y una llamada a `exit()` puede no funcionar dado el uso de threads. Para que esta aproximación funcione es necesario registrar también las funciones que cierran apropiadamente el thread de captura utilizando la función `at_exit()`. Otra posibilidad puede ser devolver un valor si la función tuvo éxito. Esta opción es todavía peor ya que la responsabilidad de cerrar el thread de exposición si la función falla, recaería en el usuario del driver que nada debe saber de él.

4.11. Los comandos de guiado

Si algo diferencia una webcam común de una cámara como la QHY5T es la posibilidad que incluye esta última de enviar comandos ST-4 a una montura robotizada para hacer guiado (Ver Apéndice A).

Utilizando la función `ctrl_msg()` se implementaron dos funciones relacionadas con los comandos de guiado, una para iniciar un pulso de guiado y otra para cancelar todo movimiento. Para la función `qhy5t_timed_move()` se indica la dirección y la duración del pulso de guiado. La dirección se establece utilizando las macros exportadas en `qhy5t.h` que pueden verse en la figura 4.11. Estas macros están definidas de manera tal que puedan componerse utilizando operaciones de bits como se hace comúnmente en C, de forma que se permiten movimientos en diagonal. Por ejemplo, para mover al

Sureste, `direction` debe ser igual a `QHY_SOUTH|QHY_EAST`. El parámetro `duration` (ver figura 4.12) expresa el tiempo del pulso de guiado en milisegundos.

```
#define QHY_EAST  0x10
#define QHY_NORTH 0x20
#define QHY_SOUTH 0x40
#define QHY_WEST  0x80
```

Figura 4.11: Macros para el parámetro de dirección

```
int qhy5t_timed_move(qhy5t_driver *qhy5t, int direction, int
    duration_msec){
  int16_t duration[2] = {0, 0};
  int cmd;

  if (! (direction & (QHY_NORTH | QHY_SOUTH | QHY_EAST | QHY_WEST))) {
    fprintf(stderr, "No direction specified to qhy5t_timed_move\n");
    return 1;
  }

  if (duration_msec > 0){
    switch (direction){
    case QHY_NORTH:
    case QHY_SOUTH:
      duration[1] = duration_msec;
      break;
    case QHY_WEST:
    case QHY_EAST:
      duration[0] = duration_msec;
      break;
    }
  }
  cmd &= direction;
  return ctrl_msg(qhy5t->handle, WRITE, 0x10, 0, cmd, (char *)&
      duration, sizeof(duration));
}
```

Figura 4.12: Función que implementa el envío de pulsos de guiado

La función anterior puede iniciar un pulso de guiado de varios segundos en alguna dirección, el cual puede ser cancelado invocando la la función `qhy5t_cancel_move()`, la cual está implementada como se muestra en la figura 4.13.

```
int qhy5t_cancel_move(qhy5t_driver * qhy5t){
  uint16_t ret;
  int cmd = 0x18;

  printf("Cancel guiding command requested\n");
  return ctrl_msg(qhy5t->handle, READ, 0, 0, cmd, (char *)&ret, sizeof
      (ret));
}
```

Figura 4.13: Función para cancelar un pulso iniciado

Capítulo 5

El viewer

Si bien el controlador para la cámara incluye una opción para compilar un programa de línea de comandos que permite configurarla y posteriormente grabar los frames en archivos en algún medio de almacenamiento, este programa no provee ningún mecanismo de visualización por ejemplo, para la etapa de enfoque. Es decir que si bien es posible indicar los parámetros de captura para una determinada secuencia de cuadros, no es posible visualizarlos hasta que el programa termine y no es sino a través de alguna herramienta externa que permita la visualización de archivos pgm.

Para mejorar la usabilidad se diseñó una herramienta que permitiera visualizar los frames antes de guardarlos, haciendo uso del driver desarrollado en este proyecto, además de la librería SDL (Simple DirectMedia Layer).

Dicha herramienta recibe el nombre de *qhy5tviewer* y constituye el segundo desarrollo más importante de este trabajo, además del driver para la cámara propiamente dicho. Este programa de línea de comandos permite visualizar el buffer capturado mediante el uso *surfaces SDL*. A grandes rasgos, el programa permite configurar un determinado modo de captura fijando los parámetros para la sesión. Luego se configura una superficie SDL que permite mostrar los cuadros, dada la configuración de dicha sesión, en la cual se irán mostrando de a uno los frames capturados. Al ser un entorno SDL, es posible capturar eventos de teclado e iniciar acciones como guardar los frames capturados en archivos, mostrar un retículo, incrementar la ganancia o enviar comandos de guiado a la montura.

Se detallan a continuación las secciones más importantes del programa *qhy5tviewer*.

5.1. La estructura general

El programa *qhy5tviewer* está escrito en lenguaje C89 al igual que el driver para la cámara QHY5T y está basado en parte de la implementación de la función `main()` del driver para GNU/Linux para la cámara QHY5 escrito por Geoffrey Hausheer. Al ser un programa de línea de comandos, cobra muchísima importancia los parámetros pasados a `main()`, ya que una vez definidos estos es que se pueden crear las estructuras relacionadas con la librería SDL entre otras. Muchas de estas configuraciones iniciales no serán cambiadas hasta que el programa haya terminado y definen para este trabajo, una sesión.

La función `main()` declara un conjunto de variables, muchas de las cuales tienen asociados valores por defecto. Por ejemplo, el tamaño del frame por defecto es 800x600, por ello tanto a la variable `width` como `height` se les asignan valores al momento de su declaración como puede verse en la figura 5.1.

```
int width = 800;
int height = 600;
int offw = (2048 - width) / 2;
int offh = (1536 - height) / 2;
int bin = 1; //binmode default 1x1
int bpp = 8;
int hblank = 142;
unsigned int vblank = 25;
unsigned int gain = 1;
unsigned int etime = 100;
char basename[256] = "image";
int debug=0;
int crossair=0;
unsigned int angle=0;
int write=0;
```

Figura 5.1: Valores por defecto al declarar las variables en `main()` de `qhy5tviewer.c`

Algunos de estos valores pueden ser cambiados luego por medio de argumentos a main(), los cuales son procesados utilizando la librería *getopt*. Las opciones de configuración que admite el programa *qhy5tviewer* se describen en la sección 5.2 de éste capítulo.

Luego de procesados los argumentos se inicializan las estructuras de SDL de acuerdo a las opciones configuradas. Por ejemplo, si se configura un tamaño de frame de 1024x768, es necesario crear una surface SDL donde estas imágenes quepan para que puedan ser mostradas de forma correcta durante la ejecución del programa.

Desde `main()`, posteriormente se invocan a las funciones de configuración del hardware de la cámara exportadas por el driver y se inicia el thread de captura.

Luego se entra a un bucle cerrado (el cual finaliza por un evento SDL) en el cual en primer lugar se procesan todos los eventos de teclado de SDL, mediante los cuales es posible por ejemplo, mostrar u ocultar el crossair (ver la sección 5.6), iniciar o detener la escritura de frames a archivos, enviar comandos de guiado a la montura o salir del programa.

Posteriormente se hace un requerimiento al driver para obtener el último frame capturado, el cual es guardado en un buffer. Los datos de este buffer son debayerizados (ver la sección 5.5) luego, para permitir mostrarlos en la surface SDL en color. Si bien el bayerizado no es óptimo en cuanto a calidad de imagen, si es lo suficientemente rápido para procesar un frame a máxima resolución entre dos capturas consecutivas de la cámara.

Luego que el frame debayerizado es mostrado en la surface SDL, se escribe en un archivo el frame sin debayerizar (si la escritura en archivos está activada). Dado que se guardan los frames RAW en lugar de los frames debayerizados, es posible utilizar mejores algoritmos de debayerización que el implementado en este trabajo a partir de los archivos guardados, en una etapa posterior de procesamiento de las imágenes.

En resumen, el esqueleto de la aplicación puede verse con el siguiente esquema.

```
declarar las variables
procesar los parámetros
inicializar las estructuras SDL
programar la cámara
iniciar la captura
while (True):
  procesar los eventos de teclado SDL
  realizar un requerimiento de un frame al driver
  debayerizar el frame
  mostrar el frame en la surface SDL
  guardar el frame en un archivo (si esto es requerido)
```

5.2. Las opciones de configuración

Desde la línea de comandos, es posible configurar los parámetros de captura de la cámara enviando argumentos a main(). Las opciones de configuración permiten ajustar el alto y ancho de la ventana de captura, la ganancia, el tiempo de exposición, el nombre para los archivos de salida y su formato, el binning y el crossair.

Para procesar los argumentos se utilizó la librería *getopt*. Para ello es necesario crear un arreglo de estructuras *option*, el cual describe cuales son las opciones y los argumentos a dichas opciones que el programa admite.

El arreglo, que debe terminar con cuatro campos en 0, está declarado como se muestra en la figura 5.2

```
struct option long_options[] = {
  {"exposure" ,required_argument, NULL, 't'},
  {"gain", required_argument, NULL, 'g'},
  {"binning", required_argument, NULL, 'b'},
  {"vblank", required_argument, NULL, 'k'},
  {"width", required_argument, NULL, 'x'},
  {"height", required_argument, NULL, 'y'},
  {"debug", required_argument, NULL, 'd'},
  {"file", required_argument, NULL, 'o'},
  {"help", no_argument , NULL, 'h'},
  {"fits", no_argument , NULL, 'F'},
  {"crossair", no_argument , NULL, 'X'},
  {0, 0, 0, 0}
};
```

Figura 5.2: Estructura con los parámetros para getopt

Cada estructura tiene cuatro miembros los cuales son

- El nombre largo de la opción, el cual es equivalente al nombre corto pero que se invoca con dos guiones en lugar de uno. Por ejemplo, invocar al programa con `-h` o con `--help` tiene exactamente el mismo resultado.
- Un número (0, 1 o 2) el cual indica si la opción no tiene argumentos, si tiene un argumento requerido o si tiene un argumento opcional. Por claridad, se utilizan las macros `no_argument`, `required_argument` o `optional_argument` las cuales provee la librería y expanden a 0, 1 o 2 respectivamente.
- Un flag que indica como se devuelven los valores para las opciones largas.
- El valor a devolver si la opción se encuentra.

A través de esta librería el programa *qhy5tviewer* acepta una serie de argumentos a `main()` que permiten variar el comportamiento del programa y modificar los parámetros de captura de la cámara. Las opciones soportadas son:

- `-t` o `--exposure`: configura el tiempo de exposición en milisegundos. Por defecto, el tiempo de exposición es de 100 milisegundos y puede variar entre 0 y 60000. Con tiempos de exposición muy bajos, inferiores a 5 milisegundos, se notaron algunos artefactos en los frames, como bandas horizontales con cambios de brillo[1]. Con tiempos de exposición muy elevados (mayores a 10000) se nota un aumento considerable en el ruido de la imagen obtenida, incluso con la ganancia en 1.

[1] Se ha notado el mismo comportamiento en circunstancias similares utilizando la cámara con el programa QGVideo5T en Windows.

- `-g` o `--gain`: configura la ganancia del sensor. Por defecto es 1 y puede variar entre 1 y 167. A mayor ganancia, también es mayor el ruido de la imagen obtenida.
- `-b` o `--binning`: configura el binning. Dado que el driver solamente soporta binning 1x1 esta opción se procesa pero finalmente se ignora.
- `-k` o `--vblank`: configura el tiempo de vertical blank y se expresa en tiempos de pixel. Por defecto es 142 y no se realizaron pruebas cambiando este valor.
- `-x` y `-y` o `--width` y `--height` : configuran el ancho y alto del frame, respectivamente. Por defecto la resolución es de 800x600 pixels. El offset es calculado para utilizar el centro del sensor.
- `-o` o `--file`: permite indicar el nombre base de los archivos a escribir. Por defecto es `image` y los archivos guardados serán `image00000.ppm`, `image00001.ppm` y así sucesivamente. El programa no controla que dichos archivos ya existan y los sobreescribe si así ocurre.
- `-F` o `--fits`: guarda los archivos utilizando el formato FITS en lugar de netpbm. El soporte para FITS hace uso de la librería *cfitsio*. En cualquier caso, las imágenes guardadas provienen del buffer leído desde el sensor y no se modifican de ninguna manera.
- `-X` o `--crossair`: Muestra el crossair sobre las imágenes visualizadas. Por defecto está deshabilitado. También puede habilitarse o deshabilitarse mediante comandos de teclado.
- `-d` o `--debug`: habilita la salida de depuración.
- `-h` o `--help`: muestra la ayuda del programa.

Es posible entonces invocar al programa *qhy5tviewer*, por ejemplo como se muestra en la figura 5.3. De esta manera el programa configurará la cámara para realizar exposiciones de 55 milisegundos con una ganancia de 24 a una resolución de 640x480 pixels. Además, si se habilita la escritura de archivos mediante comandos de teclado, estos se guardarán con los nombres `jupiter00000.ppm`, `jupiter00001.ppm` y así sucesivamente.

```
user@localhost:~$ qhy5tviewer -o jupiter -g 24  -t 55 -x 640 -y 480
```

Figura 5.3: Invocación al programa *qhy5tviewer* desde una terminal en GNU/Linux

Si bien todas estas opciones no cubren todas las posibilidades de configuración que la cámara ofrece, son más que suficientes para utilizarla para fotografía lunar, solar o planetaria haciendo foco primero y capturando las imágenes luego, como cámara de guiado (manual) o como ocular reticulado para poner la montura en estación.

5.3. El ambiente SDL

El programa *qhy5tviewer* hace uso de la librería SDL tanto para mostrar la previsualización de las imágenes que la cámara está tomando, como para capturar eventos de teclado que permiten enviar comandos al programa durante la sesión en curso. Los comandos de teclado que soporta el programa se describen en detalle en la siguiente sección.

SDL o *Simple DirectMedia Layer* por sus siglas en inglés, es una librería de funciones para el desarrollo de videojuegos en dos dimensiones, que permite entre otras cosas gestionar la carga y visualización de imágenes y detectar y administrar eventos de teclado, mouse y joystick.

La programación utilizando esta librería es sencilla. Primero se debe inicializar la librería invocando a la función `SDL_Init()` indicando que subsistemas van a utilizarse o iniciando todos mediante la macro `SDL_INIT_EVERYTHING`. Luego es necesario crear e inicializar las estructuras donde se mostrarán cada uno de los frames capturados por la cámara (Ver figura 5.4). Estas estructuras se denominan en la jerga *surfaces*. Cada *surface* en SDL representa una parte dinámica de la composición final. Estas *surfaces* se inicializan invocando a la función `SDL_SetVideoMode()`, la cual recibe como parámetros la

resolución y profundidad de color, además de una máscara que indica que propiedades de la *surface* se utilizarán. Por ejemplo es posible que la *surface* utilice aceleración gráfica mediante la máscara `SDL_HDSURFACE`, pero si esta no está soportada, la inicialización falla. Para máxima compatibilidad, la aplicación utiliza una *surface* del tamaño del frame, a una profundidad de color de 24 bits RGB. Las operaciones sobre esta *surface* se realizan en el procesador de la máquina y no mediante el uso de hardware gráfico dedicado (`SDL_SWSURFACE`) y además se obliga a inicializar la *surface* con cualquier opción de las soportadas por el dispositivo de video (`SDL_ANYFORMAT`). Los datos de un buffer son escritos en la *surface* y luego combinados con el fondo haciendo *bliting* [2] para formar la imagen final.

```
SDL_Surface * frame = NULL;
SDL_Surface * screen = NULL;
SDL_Surface * xair = NULL;
SDL_Event event;
int quit=0; //Se pone a 1 para finalizar el bucle SDL
SDL_Init(SDL_INIT_EVERYTHING);
SDL_WM_SetCaption("qhy5tviewer", "");
screen = SDL_SetVideoMode( width, height, 24, SDL_SWSURFACE |
    SDL_ANYFORMAT);
if(!screen){
  printf("Couldn't set video mode: %s\n", SDL_GetError());
  exit(-1);
}
frame = SDL_SetVideoMode( width, height, 24, SDL_SWSURFACE);
if(!frame){
  printf("Couldn't set video mode: %s\n", SDL_GetError());
  exit(-1);
}
```

Figura 5.4: Inicialización de las estructuras relacionadas con la librería SDL.

En *qhy5tviewer*, la previsualización de la imagen de la cámara está compuesta por 3 *surfaces*. La *surface* principal o `screen`, la *surface* `frame` donde se cargan los datos del último frame capturado por la cámara y la *surface* `xair` donde se dibuja el crossair. En primer lugar entonces se combinan el fondo (sin datos) y un frame, luego y si está habilitado, se carga y se combina crossair sobre los datos del frame. Una vez que el cuadro está armado, se hace un llamado a la función `SDL_Flip(screen)` la cual finalmente muestra la composición en la pantalla de video.

Las *surfaces* son estructuras nativas de la librería SDL. Es posible llenar una *surface* con los datos de un frame, simplemente haciendo apuntar al miembro `pixels` de la *surface* `frame` a la posición de inicio del buffer que contiene los datos leídos de la cámara. Sin embargo, para modificar los pixels de una *surface* directamente es necesario bloquearla para que otros procesos de la librería SDL no accedan a la misma estructura. La figura 5.5 muestra como se realiza el bloqueo de la *surface* y la modificación de los pixels de la misma.

```
debdata = debayer_data(data, debdata, qhy5t);
SDL_LockSurface(frame);
frame->pixels = debdata;
SDL_UnlockSurface(frame);
```

Figura 5.5: Modificación directa de los pixels de la *surface*.

Notar también como se ve en la figura anterior, que los datos son debayerizados antes de ser mostrados. Esto permite tener una visualización en colores de los datos RAW.

[2] El *blitting* o *bit blit*, es una primitiva básica de los sistemas gráficos de dos dimensiones, que permite combinar dos mapas de bits para formar uno solo.

En la figura 5.6 se ve el flujo de control utilizado para combinar las diferentes capas que forman la previsualización final provista por la aplicación.

```
//blitting del frame de la camara
if (SDL_BlitSurface(frame, NULL, screen, NULL)){
  printf("%s\n", SDL_GetError());
}
//carga del crossair
if(crossair){
  xair = load_crossair(angle);
  if (xair != NULL){
    //bliting del crossair en el centro de la imagen
    SDL_Rect recdst = {(width/2)-150, (height/2)-150, 0, 0};
    if (SDL_BlitSurface(xair, NULL, screen, &recdst)){
      printf("%s\n", SDL_GetError());
    }
  }
  else{
    printf("Can't load the crossair\n");
    crossair = 0;
  }
}
//Una vez armada la composicion, se muestra en la pantalla.
SDL_Flip(screen);
```

Figura 5.6: Flujo de control para armar la composición de la previsualización.

5.4. Los comandos de teclado

Además de los argumentos a `main()` que permiten controlar el comportamiento del programa, *qhy5tviewer* admite comandos durante la sesión en curso. Dichos comandos se introducen mediante el teclado, cuyos eventos son capturados por la librería SDL. Algunos de los comandos funcionan como un interruptor (*toggle*) habilitando o deshabilitando una característica, por ejemplo el crossair o la escritura de archivos. Las teclas y los comandos asociados son los siguientes:

- `Q`: Sale del programa de captura.
- `S`: Inicia o detiene la captura de frames en archivos.
- `X`: Muestra u oculta el crossair.
- `F`: Habilita o deshabilita el modo de pantalla completa.
- `V`: Imprime en la consola los FPS.
- `+`: Gira el crossair 1.40625˚.
- `-`: Gira el crossair -1.40625˚.
- $\leftarrow, \rightarrow, \downarrow$ *y* $\uparrow$: Envía un comando de guiado de 100 ms en la respectiva dirección.
- `Espacio`: Cancela los movimientos de guiado.

Estos comandos se procesan en un bucle que obliga a tratar todos los eventos antes de continuar. La figura 5.7 muestra la estructura del bucle. El tratamiento del evento consiste muchas veces en activar o desactivar un flag, mientras que el tratamiento efectivo del evento se realiza posteriormente como en el caso del crossair. Otras veces se realiza el tratamiento efectivo en el bucle como en el caso de los comandos de guiado.

```
while( SDL_PollEvent( &event ) ){
  switch( event.type ){
  case SDL_KEYDOWN:
    //process online commands
    switch (event.key.keysym.sym){
    //toggle grabe frame
    case SDLK_s:
      write = !write;
      break;
    //quit viewer
    case SDLK_q:
      quit = 1;
      break;
    case SDLK_MINUS:
      crossair = 1;
      angle = (angle - 1) % 256;
      break;
    case SDLK_PLUS:
      crossair = 1;
      angle = (angle + 1) % 256;
      break;
    case SDLK_x:
      crossair = !(crossair);
      break;
    case SDLK_v:
      showfps = !showfps;
      break;
    case SDLK_f:
      if (SDL_WM_ToggleFullScreen(frame)){
        printf("Couldn't set fullscreen\n");
      }
      break;
    case SDLK_UP:
      qhy5t_timed_move(qhy5t, QHY_NORTH, 100);
      break;
    case SDLK_DOWN:
      qhy5t_timed_move(qhy5t, QHY_SOUTH, 100);
      break;
    case SDLK_RIGHT:
      qhy5t_timed_move(qhy5t, QHY_EAST, 100);
      break;
    case SDLK_LEFT:
      qhy5t_timed_move(qhy5t, QHY_WEST, 100);
      break;
    case SDLK_SPACE:
      qhy5t_cancel_move(qhy5t);
      break;
    }
  break;
  case SDL_QUIT:
    quit = 1;
    break;
  default:
    break;
  }
}
```

Figura 5.7: Tratamiento de los eventos SDL.

5.5. La debayerización

Se conoce como interpolación cromática[3] o debayerización al proceso por el cual una imagen RAW que proviene de un sensor con una matriz bayesiana o CFA (por las siglas Color Filter Array) se convierte en una imagen color. Una imagen RAW tiene un solo canal (se ve en escala de grises), pero tiene información incompleta para la imagen color de 3 canales. Cada pixel de la imagen representa el nivel de intensidad de un color (rojo, verde o azul, dependiendo de la posición del pixel) en la imagen final.

Cabe destacar que este problema es inherente a la construcción del sensor. Este problema no existe si el sensor es monocromático (es decir, si solo capta la intensidad de la luz y esta no es filtrada de ninguna manera) o en sensores que utilizan la tecnología Foveon X3[4] donde por su construcción, cada pixel es capaz de capturar información de los tres canales (rojo, verde y azul) a la vez.

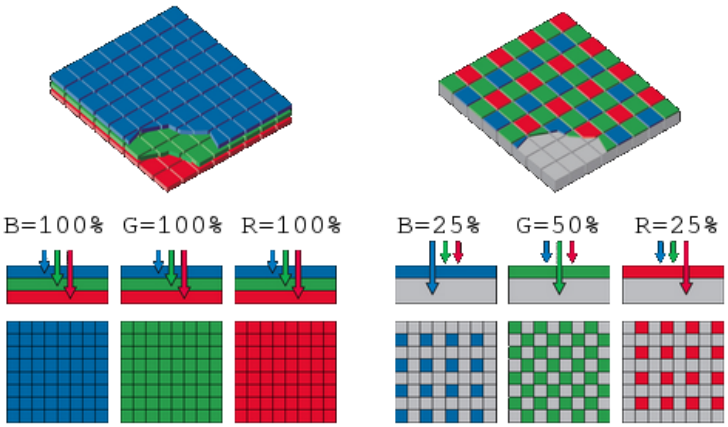


Figura 5.8: Respuesta a la luz en diferentes longitudes de onda en un sensor Foveon X3 (izq) en comparación con un sensor estándar con un CFA con patrón de Bayer (der).

Existen numerosos algoritmos que tratan el problema de la debayerización y actualmente es una campo de estudio en crecimiento, donde año a año aparecen nuevas y más sofisticadas técnicas. En "Debayer Demystified" de Craig Stark[8] se hace una comparación interesante entre cuatro de los métodos más conocidos resaltando tres de los aspectos fundamentales a tener en cuenta en la elección de uno en particular; la *Mean Square Diference* (MSE) respecto a la imagen original, la complejidad de la implementación y la complejidad computacional del algoritmo. Por ejemplo, el método *Nearest Neighbor Interpolation* es sumamente sencillo de implementar y prácticamente tiene costo cero[5] en cuanto a complejidad computacional. Sin embargo, el rendimiento del algoritmo no es bueno ya que las imágenes resultantes tienen gran cantidad de artefactos, sobre todo en las zonas de altas frecuencias (bordes), lo que se traduce en una MSE alta respecto a la imagen original. En contraste se comenta el método *Variable Number of Gradients* (VNG) que presenta una MSE muy baja (es decir, la imagen resultado es muy parecida al original) pero su costo computacional es demasiado elevado para que pueda utilizarse en aplicaciones de tiempo real.

En el medio en encuentran los métodos *Bilinear Interpolation* y *Smooth Hue Transition Interpolation* (SHTI). Ambos tienen buen rendimiento y baja complejidad computacional. La principal diferencia es que SHTI produce resultados donde se ven penalizadas las frecuencias altas en la croma, forzando transiciones suaves de color. Esto produce imágenes levemente más suaves y con menos artefactos que la interpolación bilineal.

Otros trabajos como Chang-Tan[13] o Malvar-He-Cutler[14] también presentan sus algoritmos y los comparan con otros más conocidos. Sin embargo, utilizan como métrica la relación señal-ruido pico

[3] http://es.wikipedia.org/wiki/Interpolación_cromática

[4] http://en.wikipedia.org/wiki/Foveon_X3_sensor

[5] El único costo es el asociado a la copia de cada pixel, es decir que estrictamente hablando, el orden del algoritmo es *O(n)*. Sin embargo, como no se realizan operaciones aritméticas sobre los pixels, este método es mucho más rápido que VNG a pesar que el orden de ejecución de este último también es *O(n)*.

(PSNR por las siglas en inglés para *Peak Signal to Noise Ratio*).

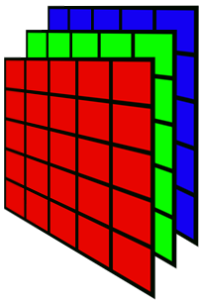


Figura 5.9: Una imagen color compuesta por 3 canales, rojo, verde y azul

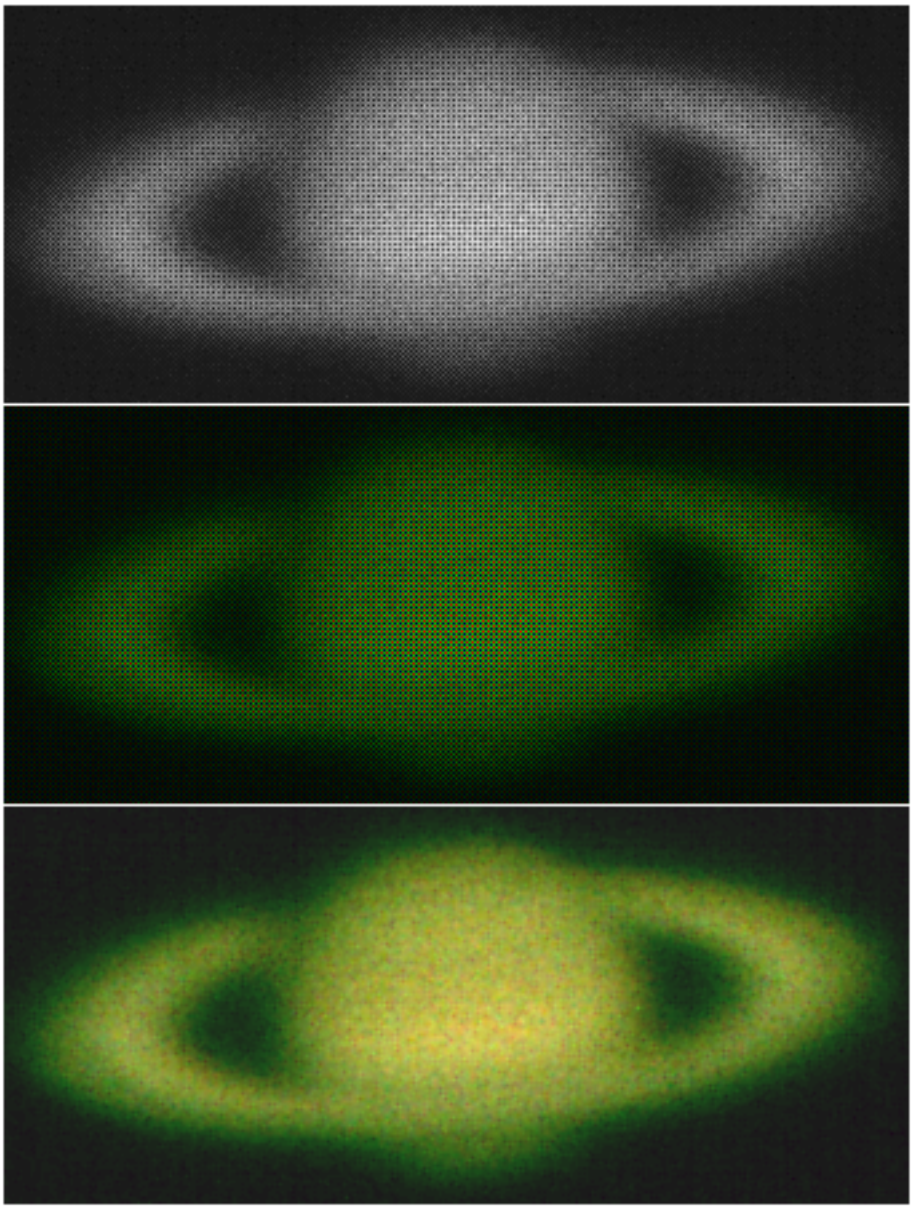


Figura 5.10: La imagen raw en escala de grises (arriba), el patrón de bayer (medio) y la imagen debayerizada por interpolación bilineal (abajo).

Dada la variedad de métodos, fue necesario elegir uno teniendo en cuenta lo siguiente:

- Las imágenes que el programa *qhy5tviewer* guarda en archivos pgm o FITS no se guardan debayerizadas. Esto permite que en una etapa de post procesado se elija el mejor algoritmo sin tener en cuenta el tiempo de ejecución.
- Es necesario mostrar imágenes claras y en color de los objetos a fotografiar, ya que el objetivo principal de mostrar lo que la cámara está fotografiando es permitir que el usuario haga foco.
- Es necesario que el tiempo de ejecución del algoritmo elegido permita mostrar imágenes en color a medida que la cámara las va entregando, en una computadora modesta.
- La sencillez a la hora de implementar el algoritmo era deseable pero no requerida.

Dados los algoritmos estudiados, las dos opciones más potables eran la interpolación bilineal y la SHTI. Para este trabajo se optó por la primera básicamente debido a que es levemente más sencilla de implementar.

5.5.1. La debayerización por interpolación bilineal

La interpolación bilineal es una forma sencilla de obtener el valor de intensidad desconocido de un pixel valiéndose de la información conocida de la intensidad de los pixels vecinos. Formalmente, sea $f(x, y) = i$ una función de la intensidad del pixel en la posición (x,y), la interpolación bilineal se define como:

$$f(x,y) \approx f(0,0)\,(1-x)(1-y) + f(1,0)\,x(1-y) + f(0,1)\,(1-x)y + f(1,1)xy.$$

También se puede definir en forma matricial de la siguiente manera:

$$f(x,y) \approx \begin{bmatrix} 1-x & x \end{bmatrix} \begin{bmatrix} f(0,0) & f(0,1) \\ f(1,0) & f(1,1) \end{bmatrix} \begin{bmatrix} 1-y \\ y \end{bmatrix}$$

En la práctica sin embargo, se tratan por separado cuatro casos, en los cuales se conoce el valor de la intensidad de un canal y se desconocen estos valores para el resto de los canales. Entonces se busca interpolar:

- Los valores de verde y azul cuando se conoce el valor de rojo.
- Los valores de rojo y azul cuando se conoce el valor de verde y los vecinos laterales son rojos.
- Los valores de rojo y azul cuando se conoce el valor de verde y los vecinos laterales son azules.
- Los valores de verde y rojo cuando se conoce el valor de azul.

Cada uno de estos casos se trata de manera diferente.

- Interpolar un pixel verde en donde se conoce el valor para el rojo o el azul. Por ejemplo, en la figura 5.11, se busca interpolar el valor de verde para la posición B4 o el de la posición C3. En cualquier caso, se cuenta con 4 vecinos para los cuales el valor de verde es conocido. Entonces, para el pixel B4, el valor del canal verde será $B4[G] = 0{,}25 * (A4 + B5 + C4 + B3)$, es decir, el promedio de los cuatro vecinos conocidos.
- Interpolar un pixel rojo/azul donde se conoce el valor para el verde. Por ejemplo, se busca interpolar el valor de rojo o azul del pixel en la posición B5. En cualquier caso se conoce el valor de rojo/azul para dos de sus vecinos. Entonces el valor de rojo para el pixel B5 será $B5[R] = 0{,}5 * (A5 + C5)$ y el valor de azul será $B5[B] = 0{,}5 * (B4 + B6)$, es decir, el promedio de los dos vecinos conocidos.
- Interpolar un pixel rojo/azul donde se conoce el valor de azul/rojo. Por ejemplo, se busca interpolar el valor de rojo en la posición B4 o el valor de azul en C3. En cualquier caso, se cuenta con 4 vecinos para los cuales el valor de azul/rojo es conocido. Por ejemplo, se busca interpolar el valor de rojo del pixel B4. Este valor está dado por $B4[R] = 0{,}25 * (A3 + A5 + C3 + C5)$, es decir el promedio de los cuatro valores conocidos más cercanos.

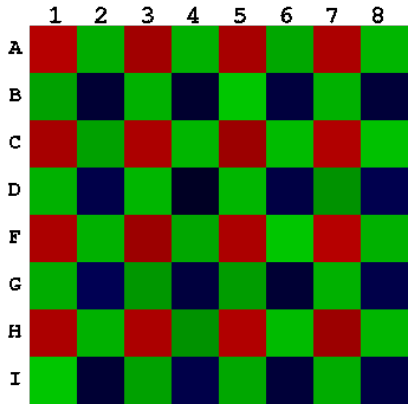


Figura 5.11

La figura 5.12 muestra un fragmento de la función `debayer_data()`. Esta función recibe un puntero a los datos de la imagen a debayerizar y obtiene las dimensiones de dicha imagen desde un puntero a una estructura `qhy5t_driver`. Las precondiciones que deben cumplirse para que esta función devuelva un puntero a una imagen debayerizada es que la imagen a debayerizar sea de 8 bits por pixel y que el patrón de Bayer sea RGGB. Esto se cumple si las imágenes a debayerizar tienen dimensiones pares.

```
for (j=1; j < h; j++){
  for (i=1; i <= w; i++){
    if (i%2 == 0 && j%2 ==0){// red pixel
      tr = *src;
      tb = 0.25*(SRCTL(src) + SRCTR(src) + SRCBL(src) + SRCBR(src));
      tg = 0.25*(SRCT(src) + SRCL(src) + SRCB(src) + SRCR(src));
    }
    if (i%2 == 1 && j%2 ==0){//green1 pixel
      tg = *src;
      tb = 0.50*(SRCT(src) + SRCB(src));
      tr = 0.50*(SRCL(src) + SRCR(src));
    }
    if (i%2 == 0 && j%2 ==1){//green2 pixel
      tg = *src;
      tr = 0.50*(SRCT(src) + SRCB(src));
      tb = 0.50*(SRCL(src) + SRCR(src));
    }
    if (i%2 == 1 && j%2 ==1){// blue pixel
      tb = *src;
      tr = 0.25*(SRCTL(src) + SRCTR(src) + SRCBL(src) + SRCBR(src));
      tg = 0.25*(SRCT(src) + SRCL(src) + SRCB(src) + SRCR(src));
    }
    src++;
    *tgt++ = (uint8_t)tb;
    *tgt++ = (uint8_t)tg;
    *tgt++ = (uint8_t)tr;
  }
}
```

Figura 5.12: Fragmento de la función `debayer_data()`

El uso de las macros `SRCT`, `SRCTR`, `SRCR`, `SRCBR`, `SRCB`, `SRCBL`, `SRCL` y `SRCTL` permite un acceso sencillo a cualquiera de los ocho vecinos de un pixel. Dado un puntero a un pixel `SRCT` retorna el vecino superior, `SRCTR` el superior derecho, `SRCBL` el inferior izquierdo y así. La figura 5.13 muestra la

declaración de estas macros. La variable `w` debe estar declarada e inicializada con el valor del ancho de la imagen. Acceder a alguno de los ocho vecinos de un pixel situado en un borde de la imagen puede devolver un pixel incorrecto (en el caso de los bordes laterales) o producir una violación de segmento (en el caso de los bordes superior o inferior). Por ello los índices de ambas sentencias `for` en la figura 5.12 evitan dichos bordes, los cuales se dejan sin tratamiento.

```
#define SRCTL(ptr) (*(ptr-w-1))
#define SRCTR(ptr) (*(ptr-w+1))
#define SRCBL(ptr) (*(ptr+w-1))
#define SRCBR(ptr) (*(ptr+w+1))
#define SRCL(ptr) (*(ptr-1))
#define SRCR(ptr) (*(ptr+1))
#define SRCT(ptr) (*(ptr-w))
#define SRCB(ptr) (*(ptr+w))
```

Figura 5.13: Declaración de las macros para acceder a los ocho vecinos de un pixel.

Fue una decisión de diseño incluir la función de interpolación cromática fuera del driver. Esta decisión es discutible. Indudablemente, la función de debayerizado depende de la disposición de los pixels en el sensor y del tamaño de la imagen a debayerizar. En la implementación para Windows del driver para esta cámara se incluyen de hecho dos funciones para debayerizar imágenes en 8 y 16 bits por pixel utilizando el algoritmo Neares Neighbor Interpolation[6]. En una posterior etapa de refactorización, las funciones de interpolación cromática deberían incluirse en el driver, implementándolas en un archivo separado para cubrir tanto los casos de 8 y 16 bits por pixel, como los casos donde las imágenes tengan dimensiones no pares.

5.6. El crossair

Durante la etapa de puesta en estación de la montura, suele ser necesario determinar la deriva de una estrella. Esto permite conocer la exactitud de la puesta en estación y da una aproximación del tiempo máximo de exposición sin guiado. Centrar una estrella en el campo de visión puede ser un proceso engañoso, ya que ni la cámara ni los oculares comunes poseen referencia alguna acerca del centro de campo de visión. Para centrar una estrella de manera efectiva es necesario utilizar un ocular reticulado[7], es decir un ocular especial que cuenta con un retículo (generalmente en forma de cruz) el cual permite conocer el centro del campo de visión con exactitud. Dichos oculares pueden conseguirse en el mercado local y si bien no son muy costosos en relación a otras piezas de equipamiento, agregan efectivamente un costo.

El programa *qhy5tviewer* permite utilizar la cámara como ocular reticulado, dibujando un retículo artificial sobre el flujo de imágenes proveniente de ésta. Otra característica que ofrece el programa es la de rotar el crossair por software, lo que permite alinearlo con los puntos cardinales sin necesidad de rotar la cámara en el portaocular del telescopio.

[6] En el driver para GNU/Linux de la cámara QHY5 no se incluyen funciones de debayerizado ya que el driver es para la versión monocromática.

[7] Un ocular es una lente intercambiable que permite variar los aumentos con los que un telescopio muestra un objeto. Generalmente, la cámara se utiliza en lugar de un ocular aunque existen configuraciones en los que el tren óptico incluye ambos, una cámara y un ocular.

Figura 5.14: Un ocular reticulado iluminado. La iluminación activa permite ver el retículo contra el fondo negro del cielo.

Cuando se activa el crossair ya sea mediante un argumento al programa principal al inicio o mediante comandos de teclado una vez que la sesión a comenzado, se ejecuta la función `load_crossair()`, la cual devuelve una surface SDL en el cual se encuentra cargada la imagen del retículo. Esta surface luego es combinada con la última imagen obtenida de la cámara utilizando la función `SDL_BlitSurface()` de la librería SDL. La función `load_crossair()` recibe como parámetro un desplazamiento que por defecto es cero y carga la imagen cuyo path es `images/crossair0.png`. El desplazamiento puede variar entre 0 y 255 e indica cual es la imagen que será cargada para el retículo. Existen por lo tanto 256 imágenes en el directorio `images/`, cada una de las cuales presenta una rotación de 1.40625 grados respecto a la anterior. Estas imágenes fueron generadas utilizando el programa de manipulación de imágenes GIMP[8]. La única particularidad respecto a estas imágenes es que tienen el fondo transparente. La función `load_crossair()` utiliza a su vez la función `IMG_Load()` que permite cargar archivos inmediatamente en surfaces SDL abstrayendo el formato de los mismos. Recibe un solo parámetro que indica la ruta relativa o absoluta al archivo a cargar. Para utilizar esta última función es necesario incluir los encabezados de la extensión de imágenes de SDL llamada *SDL_image*.

La figura 5.15 muestra la implementación de la función `load_crossair()`, mientras que la figura 5.16 muestra el fragmento de la función `main()` en el cual se carga el crossair si es necesario.

```
SDL_Surface * load_crossair(unsigned int angle){
  static SDL_Surface * crosses[256];
  char xpath[64];
  if (crosses[angle] == NULL){
    sprintf(xpath, "images/crossair%d.png", angle);
    crosses[angle] = IMG_Load (xpath);
    if (crosses[angle]==NULL){
      printf ( "Can't load image IMG_Load: %s\n", IMG_GetError());
    }
  }
  return crosses[angle];
}
```

Figura 5.15: La función `load_crossair()`. Notar la invocación a `IMG_Load()`.

[8] http://gimp.org

```
if(crossair){
  xair = load_crossair(angle);
  if (xair != NULL){
    SDL_Rect recdst = {(width/2)-150, (height/2)-150, 0, 0};
    if (SDL_BlitSurface(xair, NULL, screen, &recdst)){
      printf("%s\n", SDL_GetError());
    }
  }
  else{
    printf("Can't load the crossair\n");
    crossair = 0;
  }
}
```

Figura 5.16: Fragmento de `main()` donde se carga el crossair y se combina con la surface que contiene el frame.

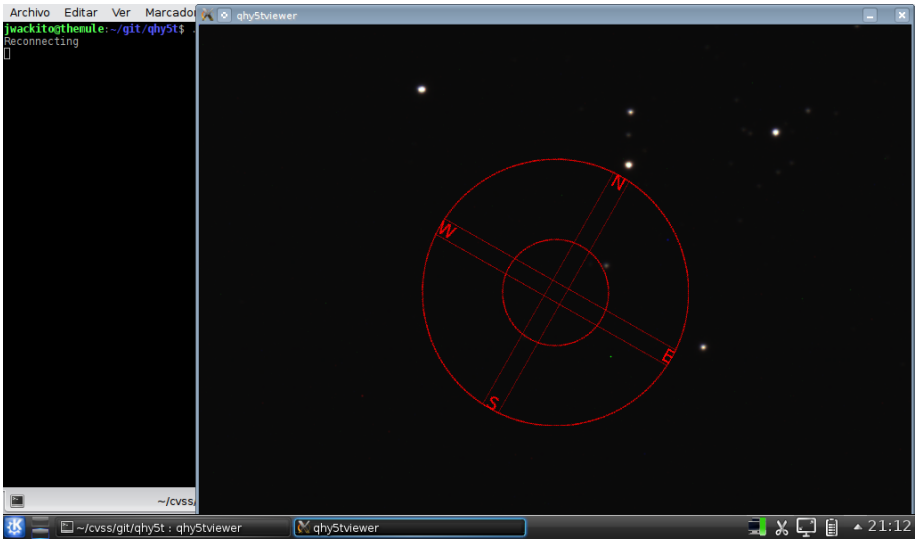


Figura 5.17: Captura de pantalla de *qhy5tviewer* utilizando el crossair.

5.7. Los archivos FITS

El formato FITS (Flexible Image Transport System)[9] es un formato estándar para el transporte e intercambio de archivos de imágenes y tablas[10] comúnmente utilizado en astronomía por profesionales y amateurs. Es un formato mucho más complejo y sofisticado que el netpbm, en cuanto soporta multitud de características que van más allá de simples imágenes. El formato FITS soporta datos unidimensionales, bidimensionales (imágenes o arreglos de datos, por ejemplo información posicional o información de distorsión en las ópticas del instrumento) y multidimensionales (por ejemplo, imágenes espectrales + datos de posicionamiento) y posee un encabezado complejo que permite definir multitud de metadatos respecto de las imágenes, como fecha, hora, tiempo de exposición, ganancia, bits por pixel y autor de la observación, entre muchos más[11].

[9] http://fits.gsfc.nasa.gov/

[10] El formato admite "imágenes" unidimensionales utilizadas en espectrometría y tablas multidimensionales de datos, por ejemplo información de posicionamiento World System Coordinate para pixels de una imagen.

[11] Puede encontrarse la lista de las *keywords* estándar y las comúnmente utilizadas en http://heasarc.gsfc.nasa.gov/docs/fcg/standard_dict.html y en http://heasarc.gsfc.nasa.gov/docs/fcg/common_dict.html respectivamente.

El programa *qhy5tviewer* permite guardar las imágenes en archivos FITS gracias a una función escrita por Giampiero Spezzano y modificada para que cumpla las necesidades de este trabajo. Sin embargo estas características son sumamente básicas y solo permiten escribir un archivo por imagen y no agrega metadata excepto la estrictamente necesaria para que el archivo tenga un formato FITS válido.

Dado que el formato FITS guarda los encabezados en formato ASCII, una simple inspección con un editor de texto revela estos campos. La figura 5.18 muestra un encabezado de datos generado por el programa *qhy5tviewer*.

```
SIMPLE  =                    T / file does conform to FITS standard
BITPIX  =                    8 / number of bits per data pixel
NAXIS   =                    2 / number of data axes
NAXIS1  =                  800 / length of data axis 1
NAXIS2  =                  600 / length of data axis 2
EXTEND  =                    T / FITS dataset may contain extensions
COMMENT   FITS (Flexible Image Transport System) format is defined in
   'AstronomyCOMMENT   and Astrophysics', volume 376, page 359;
   bibcode: 2001A&A...376..359H PROGRAM = 'qhy5tviewer'         /
   -20140317
INSTRUME= 'QHY5T camera'       /  3Mpx, 3.2x3.2 micron pixel
END
```

Figura 5.18: Encabezado de un archivo FITS generado con *qhy5tviewer*.

Recientemente se agregaron los campos PROGRAM e INSTRUMEN que indican que el archivo fue creado utilizando el programa *qhy5tviewer* y que la cámara utilizada es en efecto una QHY5T.

Para generar las imágenes en formato FITS, la función `write_fits()` hace uso de la librería *cfitsio*, la cual cuenta con funciones especificas para leer y escribir archivos en dicho formato, además de modificar todas las propiedades. Esta librería es desarrollada por el Goddard Space Flight Center (GSFC) dependiente de la agencia espacial estadounidense NASA.

La librería refleja la complejidad del formato, pero para este trabajo solo fueron necesarias algunas funciones básicas para crear el archivo de imagen, configurar los parámetros adecuados y copiar los datos de la imagen dentro del archivo FITS.

La figura 5.19 muestra la implementación de la función `write_fits()`.

```
void write_fits(void * array, qhy5t_driver * qhy5t, char *fname )
{
  //Thanks to Giampiero Spezzano who contributed this code that I
     modified.
  fitsfile *fptr;
  int status;
  unsigned int width = qhy5t->width;
  unsigned int height = qhy5t->height;
  long  fpixel, nelements;
  char filename[256] = "!"; // ! for deleting existing file and create
      new
  strncat(filename,fname,255);

  int bitpix = BYTE_IMG; /* 8-bit unsigned short pixel values       */
  long naxis = 2; /* 2-dimensional image RAW image */
  long naxes[2];
  naxes[0]=width;
  naxes[1]=height;
  status = 0;

  if (fits_create_file(&fptr, filename, &status)){
    printerror(status);
  }
  if (fits_create_img(fptr, bitpix, naxis, naxes, &status)){
    printerror(status);
  }
  fpixel = 1;                               /* first pixel to write
          */
  nelements = naxes[0] * naxes[1];          /* number of pixels to
     write */
  if (fits_write_img(fptr, TBYTE, fpixel, nelements, array, &status)){
    printerror(status);
  }
  if (fits_update_key(fptr, TSTRING, "PROGRAM", "qhy5tviewer", VERSION
     , &status)){
    printerror(status);
  }
  if (fits_update_key(fptr, TSTRING, "INSTRUME", "QHY5T camera"," 3Mpx
     , 3.2x3.2 micron pixel", &status)){
    printerror(status);
  }
  if (fits_close_file(fptr, &status)){
    printerror(status);
  }
}
```

Figura 5.19: Implementación de la función `write_fits()` en *qhy5tviewer*.

Capítulo 6

Conclusiones

6.1. Comparaciones de rendimiento

6.1.1. Impacto del debayerizado en el framerate

Las pruebas realizadas indican que a pesar que el algoritmo no es computacionalmente intensivo, tiene un gran impacto en la taza de refresco en una computadora modesta. La tabla 6.1 muestra el promedio de frames por segundo (fps) de 10 muestras a diferentes resoluciones, en el primer caso sin incluir en el ciclo la rutina de debayerización y en el segundo incluyendo la rutina de interpolación bilineal.

Resolución	Debayerizado (FPS)	Sin debayerizar (FPS)
320x240	71.96	90.73
640x480	18.97	36.05
800x600	12.38	25.98
1024x768	7.59	17.20
2048x1536	1.90	2.86

Tabla 6.1: Impacto del debayerizado en el frame rate.

Estas pruebas fueron realizadas en una Netbook Lenovo Ideapad S100 (Equipo 1) con procesador Intel Atom N570 (1.66 GHz) con 2 GB de RAM. Si bien la taza de frames por segundo es menor cuando se realiza el debayerizado, las imágenes mostradas en la sección 6.3 se realizaron utilizando este equipo, con lo que las capacidades para fotografía solar, lunar y planetaria tanto del driver como del programa *qhy5tviewer* queda demostrada, aún utilizando equipos modestos como el mencionado.

6.1.2. Comparación frente al controlador de referencia

Otras pruebas realizadas tienen que ver con la comparación de la tasa de refresco entre el driver desarrollado en este trabajo y el driver de referencia de Windows. Estos tiempos se tomaron teniendo en cuenta también el tiempo de escritura a disco. Los resultados se ilustran en la tabla 6.2.

Para que las pruebas sean equiparables, se hicieron sobre la misma computadora. En ambos programas se utilizaron las resoluciones indicadas, con la ganancia más baja (1) y el tiempo de exposición fijo en 1 ms. Ambos programas funcionan de manera similar y muestran una previsualización debayerizada del cuadro a guardar. La principal diferencia radica en que el programa QGVideo5T guarda las imágenes debayerizadas. Así, las imágenes de 2048x1536 pesan algo más de 9 MiB mientras que el programa *qhy5tviewer* guarda los frames crudos (sin debayerizar) por lo que las imágenes del mismo tamaño solo ocupan algo más de 3 MiB. Esta diferencia se nota en los resultados obtenidos, sobre todo en resoluciones más grandes.

Resolución	Escritura Linux (FPS)	Escritura Windows 8 (FPS)
320x240	99.96	100.43
640x480	39.96	37.46
800x600	27.31	23.35
1024x768	19.18	16.35
2048x1536	5.88	3.3

Tabla 6.2: Comparación de tiempos de escrituras a disco para Windows y Linux.

Estas pruebas fueron realizadas corriendo en una computadora Lenovo Thinkpad Edge E430 (Equipo 2) con 8 GiB de RAM y un disco de 500 GB a 3200 RPM. En el caso de Windows, se utilizó Windows 8 con la última versión disponible del controlador para este sistema operativo, además de la última versión disponible de QGVideo5T.

Notar que las tasas de escritura a disco en la versión para GNU/Linux corriendo sobre la E430 superan en todos los casos ampliamente las tasas obtenidas con los mismos parámetros pero en la S100. Incluso en las pruebas sin debayerización se puede apreciar una mejora aunque en este caso, más leve. Esto indica que si bien el rendimiento en una computadora promedio es aceptable, el beneficio es importante en una computadora con un procesador más potente.

6.2. Ventajas y desventajas de un controlador en espacio de usuario

En el título de este trabajo se deja entrever que el controlador para la cámara no es en espacio de kernel si no en espacio de usuario. Muchos controladores para webcams por ejemplo tienen drivers en espacio de kernel que se valen de estructuras ya definidas en el mismo, como *v4l2*[1] la cual define que comportamientos debe implementar un dispositivo que capture o transmita video, *usb* que permite comunicaciones usando interfaces USB entre dispositivos o *i2c*, que permite que el kernel se comunique directamente con dispositivos que soporten el protocolo I^2C, entre otras. Este trabajo se realizó en espacio de usuario, en lugar de utilizando las estructuras ya provistas por el kernel, principalmente por que el controlador de referencia para GNU/Linux para la cámara QHY5 ya estaba escrito en espacio de usuario.

Esto representa grandes beneficios. Por un lado se ve reducida la complejidad de introducir modificaciones en el Kernel Linux, el cual, al ser un proyecto de envergadura, requiere una inversión de tiempo importante en el estudio de las estructuras de datos utilizadas, antes de que se puedan agregar modificaciones en forma natural. Otro beneficio importante es que no es necesario mantener el ritmo de actualización que impone el desarrollo del Kernel Linux, en el cual se libera una nueva versión cada 2 o 3 meses. Muchas de estas modificaciones no afectan a los controladores pero es común notar que en el transcurso de un año, todos los controladores requieren algún mantenimiento.

Una de las principales razones esgrimidas a la hora de encarar el desarrollo de controladores en espacio de kernel es el rendimiento. Al correr en espacio de kernel todas las llamadas al driver tienen prioridad lo que suele redundar en un mayor velocidad de reacción. El tope de rendimiento en cuanto a la taza de frames por segundo que soporta el dispositivo está determinada por diversos factores. Uno es el límite teórico que impone el bus USB. La taza de transferencia del bus USB, en su especificación 2.0 de alta velocidad indica una tasa pico de 30 MBytes por segundo. En imágenes de 3Mpx, esto daría una tasa de refresco de 10 fps. Otro límite importante es la tasa de escritura en el dispositivo de almacenamiento donde se guardarán las imágenes obtenidas[2]. Si por el contrario tomamos como límite las tasas de frames del controlador de referencia para las diferentes resoluciones, podremos comparar si al menos la tasa de refresco del driver desarrollado para este trabajo es equiparable a la tasa de refresco del controlador de referencia. Como se puede ver en la sección 6.1.2, dichos frame rates son

[1] Video for Linux 2 (v4l2 o también V4L2) es una API de desarrollo para controladores de dispositivos relacionados con video como cámaras, capturadoras de TV o tarjetas de Digital Video Broadcasting.

[2] De las pruebas realizadas se desprende que el limitante es la tasa de transferencia del USB, la cual es mucho menor que la tasa de escritura para un disco rígido estándar.

comparables, por lo que al menos este controlador esta a la altura de rendimiento del controlador de referencia.

Por otra parte, los controladores en espacio de kernel exportan dispositivos al sistema de archivos, por lo que es posible que los programas de espacio de usuario accedan a las funciones exportadas por el controlador del dispositivo en cuestión por medio de dicho archivo (en general, los archivos que representan dispositivos se encuentran en el directorio `/dev`). Dado que esta interfaz es uniforme, los programas pueden utilizar los dispositivos a través del uso de alguna librería en espacio de usuario que enlaza las funciones en espacio de kernel con las de espacio de usuario. Debido a esto es que existe una amplia gama de programas que haciendo uso de la API *v4l2* permiten visualizar streams de video y grabar dicho stream en archivos, como mplayer, cheese y motion, entre otros. Dado que el controlador en espacio de usuario no exporta una API *v4l2*, la cámara QHY5T no puede utilizarse con estas aplicaciones. Esta es una de las principales desventajas de tener un controlador en espacio de usuario ya que mucho software preexistente, de gran calidad y madurez, no puede utilizarse con esta cámara y es necesario adaptar dicho software o escribir nuevo para que esto último sea posible.

6.3. Avances logrados

A comienzos del año 2012, cuando la cámara fue adquirida, pudo comprobarse que a pesar que existía el soporte para el sensor en el Kernel Linux, la cámara no funcionaba en este sistema operativo. Esto se debía principalmente a que la interfaz I^2C del sensor no es expuesta por la cámara si no que la comunicación entre la computadora y el sensor se realiza a través del firmware cargado en el procesador del dispositivo.

Durante los dos años siguientes se trabajó, mediante ingeniería inversa, análisis de tráfico y estudio de los controladores de referencia, en el desarrollo de un controlador para GNU/Linux y un programa visor que permitiera grabar imágenes astronómicas, los cuales se describen en la presente tesina. El trabajo en la mejora del driver todavía continúa.

Utilizando programa *qhy5tviewer* fue posible obtener los siguientes resultados. Cabe destacar que todas las imágenes aquí presentadas fueron realizadas exclusivamente utilizando el sistema operativo GNU/Linux (en el Equipo 1, salvo que se especifique lo contrario), no solo durante la captura sino también en el post procesado de las imágenes capturadas para llegar a la versión final de cada una.

6.3.1. Resultados en fotografía solar

A continuación se muestra en la figura 6.1 una imagen cruda en formato pgm obtenida durante el mes de enero de 2014. Dicha imagen fue capturada con la cámara QHY5T utilizando el programa *qhy5tviewer* desde Lihuen GNU/Linux. La cámara estaba adosada a un telescopio newtoniano Sky-Watcher de 200 mm de apertura y 1000 mm de distancia focal (SW 200/1000), con un filtro solar Baader + un filtro Astrodon Photometrics (Johnson/Cousins) en banda V. La figura 6.2 es el resultado de procesar más de 2000 imágenes como la de la figura 6.1. Las imágenes fueron debayerizadas individualmente utilizando el script debayer.py, el cual utiliza la función de debayerización de la librería OpenCV. Las etapas de alineación, apilado y aplicación de wavelets fue realizada utilizando el programa Registax 6. Finalmente se utilizó el programa GIMP para aplicar una máscara de desenfoque y convertir el archivo .tiff resultante del procesado con Registax a formato .jpg. La cantidad efectiva de imágenes apiladas en este caso fue de alrededor de 400, mientras que el resto fueron descartadas.

Dadas las condiciones de seeing al momento de la captura, es posible que los resultados no sean los mejores posibles y que puedan resolverse más detalles con la atmósfera más clara.

Las imágenes corresponden a la mancha AR-1967, considerada una de las manchas más grandes del último ciclo solar hasta el momento de escribir este trabajo.

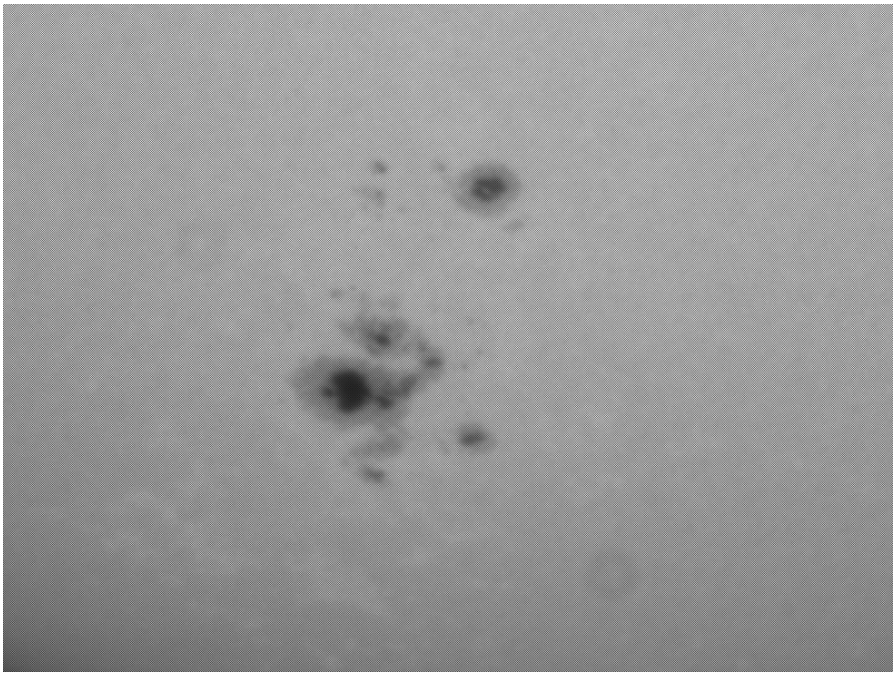


Figura 6.1: La imagen RAW obtenida con la cámara QHY5T desde una computadora corriendo Lihuen GNU/Linux.

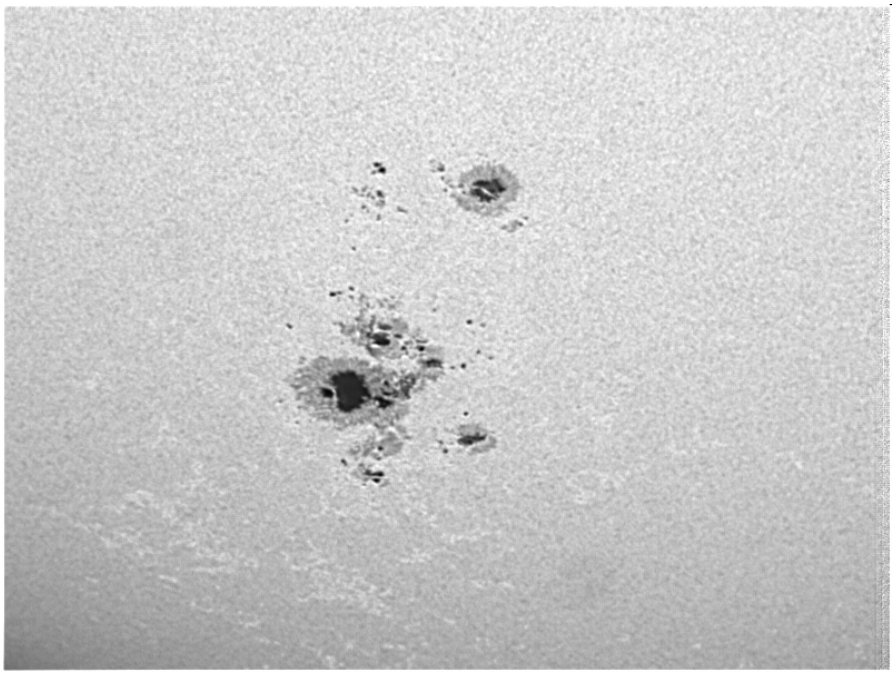


Figura 6.2: Imagen obtenida a partir de 2000 cuadros similares a los de la figura 6.1.

6.3.2. Resultados en fotografía planetaria (Júpiter)

En la figura 6.3 puede verse una imagen cruda de Júpiter tomada en enero de 2014. La fotografía fue adquirida utilizando un telescopio newtoniano de 150 mm de diámetro y 750 mm de distancia focal durante una noche de excelente seeing y transparencia. Además de la cámara se utilizó una

lente TeleVue PowerMate x2.5[3] pero no se utilizó ningún otro implemento. La figura 6.4 muestra el resultado de utilizar el programa debayer.py para debayerizar la imagen de la figura anterior.

Figura 6.3: Imagen cruda de Júpiter tomada con la cámara QHY5T.

Figura 6.4: La imagen de la figura 6.3 debayerizada con debayer.py.

La imagen de la figura 6.5 es el resultado de procesar unos 1200 cuadros crudos utilizando debayer.py, Registax 6 y GIMP. En ella puede apreciarse con claridad los cinturones ecuatoriales, festones oscuros en el cinturón norte, las bandas tropicales, la tormenta conocida como la Gran Mancha Roja (GRS, por sus siglas en inglés) y varias tormentas menores sobre la superficie del planeta. A la izquierda se ve la luna galileana Ío con su característico color rojizo.

[3] El TV PowerMate es una lente negativa similar a una lente barlow, diseñado por Al Nagler, que al alargar la distancia focal del telescopio (en este caso en 2,5 veces) permite imágenes con más aumento, aunque no con más resolución.

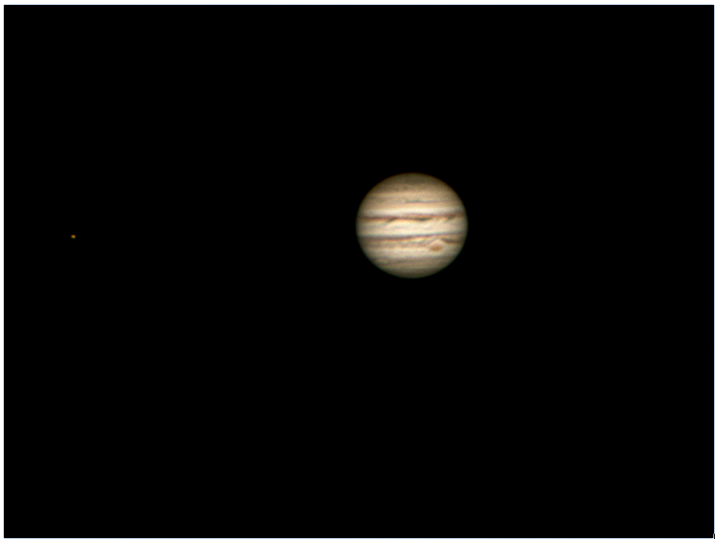


Figura 6.5: Imagen obtenida a partir de 1200 cuadros obtenidos con la QHY5T y el programa qhy5tviewer.

6.3.3. Resultados en fotografía lunar

El mes lunar tiene una duración de 29.53 días solares. En este tiempo la Luna pasa por todas sus fases. Durante la fase de Luna llena o casi llena es muy difícil hacer buenas tomas ya que al recibir luz directa del sol en un ángulo de casi 90^{o}, ninguno de los cráteres, valles o montañas tienen sombra y la falta de contraste genera imágenes lavadas sin muchos relieves aparentes. Sin embargo esta es una oportunidad de fotografiar el disco lunar completo, además de eventos asociados como el transito de satélites, aviones o aves. Sin embargo esto no fue posible dado que el sensor de la cámara no es lo suficientemente grande para que el disco quepa completo, al menos en los equipos con los que se contaba para realizar las pruebas.

Durante las fases de cuarto creciente y cuarto menguante, así como dos días antes y dos días después, la zona del terminador (el amanecer o el atardecer lunar) presenta relieves y sombras de una belleza incomparable que solo alcanzan a comprender aquellos que han puesto un ojo en el ocular. Durante la fase de cuarto creciente, se pueden fotografiar dichos relieves entre las 18 y las 22 horas, horario bastante cómodo para cualquier aficionado con necesidad de trabajar durante la semana. En la fase de cuarto menguante en cambio, solo es posible fotografiar la luna entre las 2 y las 6 de la madrugada lo que hace la tarea bastante más inconveniente en las mencionadas condiciones. Por supuesto es posible fotografiar la Luna en dicha fase durante el día, pero el contraste ofrecido por el cielo diurno es muy inferior al del cielo nocturno. Lo que deja una ventana de unos cinco días por mes en los cuales, si las condiciones del clima son apropiadas, es posible tomar algunas fotografías del los cráteres más prominentes. Demás está decir que las pruebas aquí expuestas no fueron exhaustivas dado que los días que fueron realizadas no fueron óptimos en cuanto a calidad de cielo.

En la figura 6.6 puede verse una imagen cruda de los Montes Apeninos tomada en enero de 2014. La fotografía fue adquirida utilizando un telescopio newtoniano de 150 mm de diámetro y 750 mm de distancia focal, sin seguimiento motorizado, durante una noche de buen seeing y transparencia pobre (nubes intermitentes).

Figura 6.6: Imagen cruda de los montes Apeninos tomada con la cámara QHY5T.

La figura 6.7 muestra el resultado de alinear y apilar solamente unas 400 imágenes utilizando el programa Registax 6. Notar que tanto la imagen procesada como el crudo presentan zonas sobreexpuestas. En este caso debió haberse utilizado un filtro de densidad neutra para reducir el brillo o una ganancia o tiempo de exposición más reducido.

Figura 6.7: Post procesado de unas 400 imágenes utilizando debayer.py y Registax 6

La siguiente foto es una de las primeras capturas obtenidas con la cámara. Se trata del cráter Gassendi, ubicado al centro sud este de la superficie lunar. Dicha imagen fue publicada en el sub foro de Linux del fabricante QhyCCD[4] por el autor de este trabajo y constituye una de las primeras pruebas exitosas tanto del driver como de las etapas tempranas de desarrollo visor.

[4]http://qhyccd.com/ccdbbs/index.php?topic=4021.0

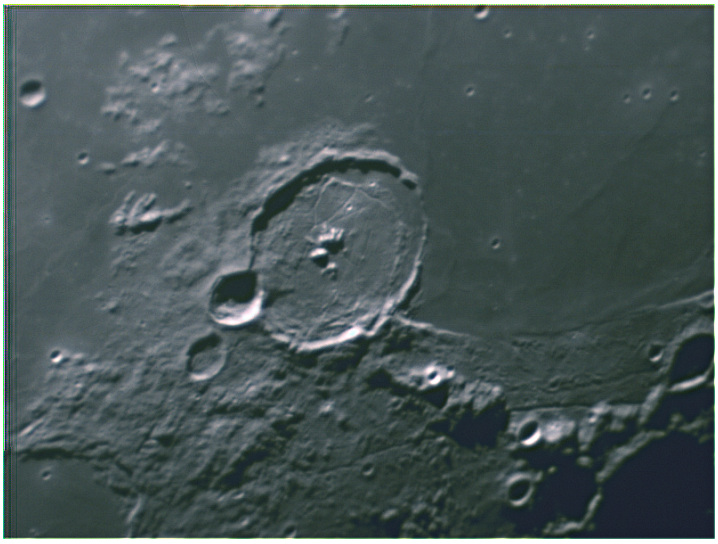


Figura 6.8: Una de las primeras imágenes capturadas utilizando la cámara QHY5T en GNU/Linux.

6.4. Resultados en fotografía planetaria (Saturno)

En la figura 6.9 se puede ver una imagen cruda de Saturno tomada en febrero de 2014. La fotografía fue adquirida utilizando un telescopio newtoniano de 200 mm de diámetro y 1000 mm de distancia focal durante una noche de buen seeing y transparencia. Además de la cámara se utilizó una lente TeleVue PowerMate x2.5. La figura 6.10 muestra el resultado de utilizar el programa debayer.py para debayerizar la imagen de la figura anterior. Además en este caso se utilizó la función de crop incluida en el programa. Si bien los frames originales son de 800x600, el programa debayer.py centra el planeta y recorta esa sección luego de debayerizar, produciendo cuadros mucho más chicos, donde el planeta se encuentra casi en el centro. Esto mejora el alineado en Registax y reduce enormemente el tiempo de procesado.

Figura 6.9: Imagen cruda de Saturno tomada con la cámara QHY5T.

Figura 6.10: La imagen de la figura 6.9 debayerizada y recortada con debayer.py.

La imagen de la figura 6.11 es el resultado de procesar 820 cuadros. Además del recorte con debayer.py, se utilizó Registax 6 para alinear y apilar los cuadros. En la etapa de apilado es posible duplicar la resolución de la cada imagen, por lo tanto el resultado es una imagen más grande. Posteriormente también se utilizó Gimp 2.8 para realzar las zonas de frecuencias altas utilizando una máscara de desenfoque. En la fotografía pueden verse claramente los anillos del gigante gaseoso, incluso el anillo C (el más interno) aunque mucho más tenue que el resto. La División de Cassini se encuentra bien resuelta en toda la circunferencia del los anillos e incluso puede apreciarse el disco planetario a través de la División. También pueden verse los diferentes colores de las franjas gaseosas del planeta. Una más clara en la zona ecuatorial, una más oscura en la zona tropical y por último el Vórtice Hexagonal Polar Norte, aunque este último no se halla bien resuelto[5].

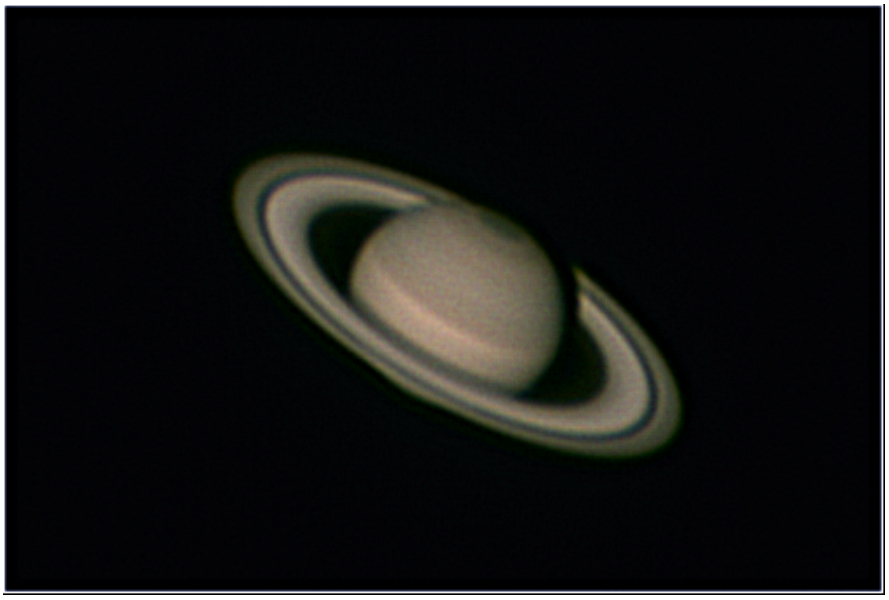


Figura 6.11: Imagen obtenida a partir de 820 cuadros obtenidos con la QHY5T y el programa qhy5tviewer, procesados con Registax 6.

[5] A mayores aperturas, es posible ver claramente que este vórtice tiene una extraña forma hexagonal. Fue observado por primera vez por la sonda Voyager en 1981. Más en http://en.wikipedia.org/wiki/Saturn%27s_hexagon

Capítulo 7

Trabajo a futuro

A pesar que el trabajo cumple con los objetivos propuestos, muchas son las mejoras que se pueden practicar a lo aquí presentado. Aquí se exponen algunas sugerencias para trabajos futuros pero esta lista no es ni por mucho exhaustiva. La astronomía amateur es un campo de juegos para los estudiantes de informática, lleno de desafíos atrapantes.

7.1. El soporte para imágenes de 16 bits

Una mejora importante para este trabajo sería incluir el soporte para imágenes de 16 bits por pixel. Exige modificar varias funciones, una de ellas está relacionada con la cantidad de datos que son transferidos durante la lectura de un frame. También es necesario modificar el tamaño de los buffers y la función que mapea el frame en una imagen. Dado que el ADC devuelve 10 bits, estos son transferidos desde el dispositivo como 2 bytes, donde el byte menos significativo es transferido primero. Será necesario escribir una función que haga el intercambio o los reordene una vez que los datos están en el buffer. Además será necesario reescribir la función de debayerizado para tener en cuenta la mayor profundidad de color. Se estima que esta mejora tiene una dificultad media.

7.2. El soporte para binning 2x2

Este cambio permitiría aprovechar la función de binning de 2x2 que provee el sensor, disminuyendo el ruido a costa de disminuir la resolución de la imagen final. Se debe modificar la función qhy5t_program_camera() ya que los valores para los registros en modo binning 2x2 cambian sustancialmente. Además es necesario revisar el tamaño de los buffers y la cantidad de datos a transferir en las lecturas de datos desde el sensor. Se estima que esta modificación tiene una dificultad alta debido al análisis que requiere.

7.3. Migración a libusb-1.0

Al momento de comenzar este trabajo, la librería libusb-1.0 estaba en etapa de pruebas. No fue hasta julio de 2013 que fue liberada la actual versión estable. La migración a libusb-1.0 sin embargo no plantea grandes desafíos ya que las comunicaciones entre la cámara y el driver se encuentran centralizadas en la función `ctrl_msg()` para mensajes de control y en `exposure_thread()` en la cual se leen los datos del sensor. A pesar de esto, puede ser necesario que cambien algunos tipos ya que `usb_dev_handle` no existe en la nueva versión de la librería.

7.4. Compatibilidad con herramientas preexistentes

Existen diversas herramientas a las que podría integrarse el trabajo realizado. Una de ellas es el nuevo SDK para GNU/Linux desarrollado por la empresa QhyCCD. Este SDK actualmente soporta las cámaras del fabricante cuyos modelos son posteriores a 2012 como por ejemplo las series QHY6,

QHY21, QHY23 y QHY5-II, entre otras. Sin embargo, al momento de escribir este trabajo, dicho SDK no soporta el modelo QHY5T. Dado que el SDK está liberado utilizando una licencia GPL2, es posible desarrollar una capa de abstracción entre la API del SDK y las funciones implementadas en el driver que cubre este trabajo. Esto permitiría que la cámara QHY5T funcione con herramientas que ya están siendo desarrolladas utilizando dicho SDK como *OpenSkyImager* desarrollado por Giampiero Spezzano.

Otra herramienta interesante parece ser *lin_guider*. Esta herramienta ha sido especialmente diseñada para realizar guiado, es decir que una vez calibrado puede detectar la dirección de la deriva de una estrella usando información de el cuadro actual y los anteriores y enviar comandos de guiado a la montura para mantenerla apuntando siempre al mismo lugar. Este programa actualmente soporta una variedad de cámaras de la empresa QhyCCD además de cámaras de otros fabricantes. Sin embargo al momento de escribir este trabajo, la cámara QHY5T no está soportada.

Para esta integración puede ser necesario reescribir y modificar la API del driver desarrollado en este trabajo para que se ajuste a la interfaz de desarrollo del SDK y de *lin_guider*. Dado que ambos están escritos mayormente en C++, puede requerir una inversión de tiempo y esfuerzo considerable.

7.5. Substracción de dark frames automática

La substracción de dark frames es una de las técnicas principales para reducir el ruido en imágenes astronómicas. La técnica consiste en tomar entre 10 y 30 imágenes con la tapa del telescopio puesta (o con el obturador cerrado, en aquellas cámaras con obturador físico) con exactamente los mismos parámetros que para la captura del objeto que se desea fotografiar. Estos cuadros negros son conocidos en la jerga astrofotográfica como dark frames y representan el ruido térmico del sensor, es decir ruido aleatorio producido por un fenómeno conocido como dark current[1]. Promediando los dark frames se puede obtener un promedio del ruido del sensor el cual puede ser substraído de los light frames (los cuadros donde aparece el objeto fotografiado). Tanto el promedio como la substracción de imágenes se pueden realizar haciendo operaciones aritméticas entre pixels. Dada la baja complejidad del algoritmo, es posible hacerlo en tiempo real sin que se vean disminuciones en el frame rate. Se estima que esta modificación es de dificultad baja.

7.6. Modificación de parámetros de captura durante la sesión

Para que el programa qhy5tviewer permita cambiar las configuraciones de la cámara durante la sesión se requiere que la información sea ingresada de forma cómoda. Por ejemplo, cambiar la resolución de las imágenes capturadas requiere que se puedan ingresar dos números, por ejemplo utilizando un widget[2]. SDL no tiene estos elementos en forma nativa como otras librerías gráficas como Qt, GTK o WXWidgets.

Migrar la aplicación qhy5tviewer a alguna de estas librerías implica un rediseño y probablemente una reescritura completa de la misma.

Otros cambios pueden ser más sencillos. Por ejemplo incrementar o decrementar el tiempo de exposición o la ganancia puede involucrar solamente agregar algunos comandos SDL a los que ya han sido implementados(ver sección 5.4).

7.7. Mejoras a la función write_fits

Actualmente la función write_fits() no incluye metadatos importantes en el encabezado del archivo. Algunos datos como fecha y hora de la captura, tiempo de exposición, lugar de observación, etc pueden ser deseables si se van a utilizar los archivos con fines científicos. Mientras que los metadatos acerca de tiempo de exposición o nombre del observador pueden manejarse como cadenas de caracteres simples, el lugar de observación, las coordenadas del objeto o las fechas requieren tipos específicos provistos

[1] Dark current o corriente de oscuridad es una respuesta constante pero aleatoria que exhiben los receptores de radiación (en el caso de un sensor fotográfico, radiación lumínica) durante los períodos en los cuales no está recibiendo radiación alguna. Este efecto, en general es más intenso a medida que la temperatura del sensor aumenta.

[2] Un widget es un elemento de programación en interfaces de usuario gráficas, como un botón, un cuadro de dialogo o un elemento de entrada de texto.

por *cfitsio*, para que el archivo generado sea compatible con los visores disponibles. No es trivial aprender a manejar estos tipos de datos y puede requerir una investigación importante relacionada con el formato, pero no se espera que esta tarea represente un trabajo de complejidad alta.

7.8. El instalador para Debian

Tanto el driver como el visor no cuentan con un instalador y por ahora se distribuyen los archivos fuente con instrucciones para compilar y construir el visor. Los usuarios finales menos técnicos se verían enormemente beneficiados por un paquete que permita instalar tanto los drivers como el visor utilizando las herramientas provistas por su distribución de GNU/Linux.

Una de las distribuciones de GNU/Linux más difundidas sin duda es Debian, la cual tiene muchas distribuciones derivadas entre las cuales se encuentran Ubuntu y derivados, Linux Mint y por supuesto Lihuen GNU/Linux, la distribución desarrollada en el Laboratorio de Investigación en Nuevas Tecnologías en Informática de la Facultad de Informática de la Universidad Nacional de La Plata.

No es una tarea sencilla crear un paquete para Debian GNU/Linux que cumpla con los estándares para que pueda ser agregado a los repositorios oficiales. Afortunadamente la documentación al respecto es basta[3][4].

Mientras que desarrollar el paquete y hacerlo disponible para descarga desde repositorios de terceros puede ser una tarea sencilla, el proceso de admisión como paquete oficial para Debian GNU/Linux puede tomar bastante tiempo.

7.9. Obtención del código

Para encarar cualquiera de las tareas anteriores es necesario contar con el código fuente del trabajo que se presenta en esta tesina. Dicho código se adjunta como archivos en los medios digitales requeridos para la presentación, pero además puede descargarse desde internet no solo la versión presentada en este trabajo sino toda la historia de desarrollo del proyecto.

Tanto el código del controlador, como el del visor pueden descargarse desde github.com. Además se encuentran disponibles el firmware el cual se distribuye con permiso del desarrollador "así como está" y sin garantías en formato IHEX y las reglas de *udev* para cargar el firmware a la cámara con *fxload* automáticamente cuando se detecta que se conectó una cámara QHY5T al bus USB.

En distribuciones basadas en Debian, puede instalarse *GIT* con la herramienta de instalación *apt*. Una vez instalado GIT, solo es necesario clonar el repositorio, lo cual se puede lograr mediante el siguiente comando

```
user@host:~$ git clone https://github.com/jwackito/qhy5tviewer.git
```

Una vez clonado el repositorio es posible ver el código de esta tesina haciendo

```
user@host:~$ git checkcout version0.3
```

7.10. Compilación del controlador y el visor

Una vez obtenido el código, es posible construir la aplicación qhy5tviewer mediante el script compile.sh. Para que compile sin problemas es necesario instalar las librerías de las que el código depende, a saber libpthread, libsdl-1.2, libsdl-image1.2, cfitsio y libusb-0.1. Estos paquetes se pueden instalar en Debian.

En distribuciones basadas en Debian Wheezy, esto puede lograrse mediante el siguiente comando ejecutado como superusuario:

```
user@host:~# apt-get install build-essentioal libsdl-image1.2-dev libcfitsio3-dev libusb-dev
fxload
```

Si bien el paquete fxload no es necesario para compilar el paquete, si es necesario para cargar el firmware a la cámara una vez que esta se conecta a algún puerto USB. Es recomendable copiar el contenido de la carpeta `etc/` provisto en el código a la carpeta `/etc` del sistema. Esta carpeta contiene

[3] http://www.debian.org/devel/

[4] http://www.debian.org/doc/manuals/maint-guide/

el firmware y los archivos de configuración de *udev* para que la carga de dicho firmware se produzca de manera automática cuando se detecta un dispositivo QHY5T compatible.

Puede ser de mucha ayuda suscribirse al foro de QhyCCD[15] de desarrolladores para GNU/Linux, en el cual suelen discutirse tópicos relacionados con el desarrollo de drivers y programas para los dispositivos de este fabricante.

Capítulo 8

Apendice A

8.1. Sobre las técnicas de guiado

Para la fotografía de objetos de espacio profundo como galaxias y nebulosas, es necesario como se mencionó anteriormente, realizar tomas con cierto tiempo de exposición. Estos tiempos pueden variar de varios segundos a varios minutos, dependiendo del brillo del objeto y el o los filtros utilizados.

Sin embargo, cuanto mayor es el tiempo de exposición, mayor es el movimiento aparente del cielo nocturno. Para contrarrestar este movimiento y así evitar que las estrellas dejen trazos en la capturas en lugar de aparecer como objetos puntuales, los aficionados utilizan monturas ecuatoriales motorizadas, las cuales una vez puestas en funcionamiento permiten realizar tomas de entre 1 y 3 minutos, dependiendo de la precisión a la hora de poner la montura en estación. Para tiempos de exposición mayores a los 3 minutos, sin embargo, es necesario recurrir a otras técnicas ya que no solo la puesta en estación influye en la calidad del seguimiento celeste, sino también el error periódico introducido por los engranajes de la montura, la flexión del tubo principal sobre la montura o incluso el flujo de corriente irregular pasado a los motores[16].

Una de las técnicas más utilizadas consiste en utilizar un segundo telescopio, generalmente más pequeño, al cual se conecta la cámara guía. Tanto la montura como la cámara deben tener soporte hardware para poder realizar el guiado. Generalmente, la cámara se conecta a la montura directamente a través del puerto de guiado que implementa el protocolo ST-4. La cámara además se encuentra conectada a una PC que corre el software de guiado. La cámara envía imágenes al software de guiado. Se selecciona una estrella guía en el campo cercano al objeto que se quiere fotografiar y se calibra el software de guiado para determinar la dirección de la deriva. Una vez calibrado el software, el guiado se realiza en forma automática. El software analiza las imágenes que genera la cámara guía y determina si la estrella deriva. Si es necesario realizar alguna corrección, el software de guiado invoca a una rutina del driver de la cámara para que esta le envíe los pulsos de guiado necesarios a la montura para mantener la estrella centrada.Un ejemplo de software de guiado que funciona en GNU/Linux es el *lin_guider*[1].

En la figura 8.1 puede verse una configuración típica para realizar guiado. Sobre el telescopio principal (un SkyWatcher 200/1000, en negro) se ve montado un telescopio más pequeño, en este caso, un tubo guía SkyWatcher 80/400 (en color azul) con una cámara guía QHY5. Todo el equipo está montado sobre una montura SkyWatcher HEQ5 con soporte para el protocolo ST-4.

[1] http://sourceforge.net/projects/linguider/

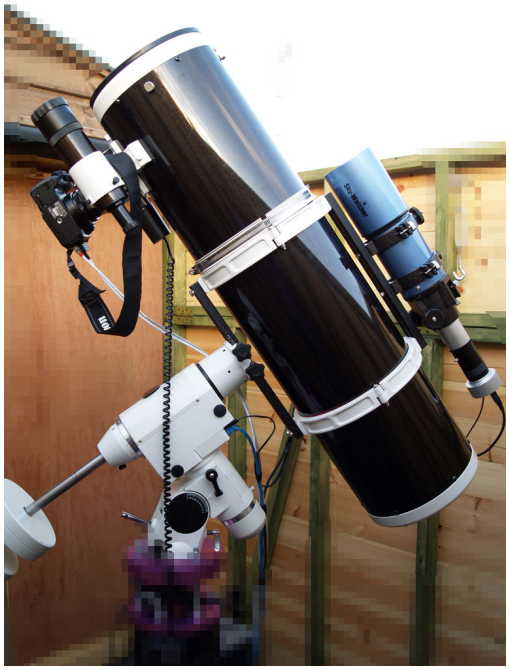


Figura 8.1: Configuración para realizar fotografías de larga exposición con guiado.

8.2. Sobre las técnicas de procesado para fotografía lunar y planetaria

Para obtener imágenes como la de la figura 8.2 son necesarios varios pasos. La técnica consiste en tomar varias fotografías del mismo objeto. Estas imágenes luego son registradas, de manera que el objeto a fotografiar siempre se encuentre en la misma posición. Existen varios algoritmos para registrar imágenes, incluso algunos que logran precisión sub-pixel. Muchos de ellos vienen implementados en los programas de procesado. Posteriormente las imágenes son apiladas, proceso que consiste en obtener una única imagen, donde la intensidad del pixel en la posición (x,y) se calcula utilizando los valores de pixels de la posición (x,y) de todas las imágenes. Suele utilizarse el promedio, la media y la mediana, aunque existen otros algoritmos como "Sigma clipping"que pueden utilizarse en escenarios específicos. Este proceso elimina el ruido y aumenta la señal del objeto a fotografiar. Una vez aumentada la señal, se puede aplicar diferentes filtros para realzar los detalles, como transformadas de Wavelets[2], Wiener [3] para reducción de ruido, filtrado en frecuencia para eliminar bandas, alinear canales RGB, ecualización de histograma, etc.

Registax 6 ofrece una entorno para realizar las etapas de registro y apilado, además de algunos filtros sencillos, como wavelets, ecualización de histograma y alineación de canales, además de algunos filtros específicos más complejos como "Derotation". Este filtro detecta y compensa el movimiento alrededor del eje de rotación y es sumamente útil en Júpiter. Júpiter tiene un período de rotación de 9 hs, con lo cual, capturar imágenes durante más de 45 segundos produce resultados borrosos a grandes aumentos, ya que es imposible alinear las primeras imágenes con las últimas. Gracias al filtro "Derotation", es posible sin embargo obtener imágenes detalladas a partir de varios minutos de captura. Fotógrafos como Demian Peach[17] y Christopher Go[18] hacen uso de esta misma técnica, aunque ambos con resultados sumamente diferentes.

[2]http://en.wikipedia.org/wiki/Wavelet_transform
[3]http://en.wikipedia.org/wiki/Wiener_filter

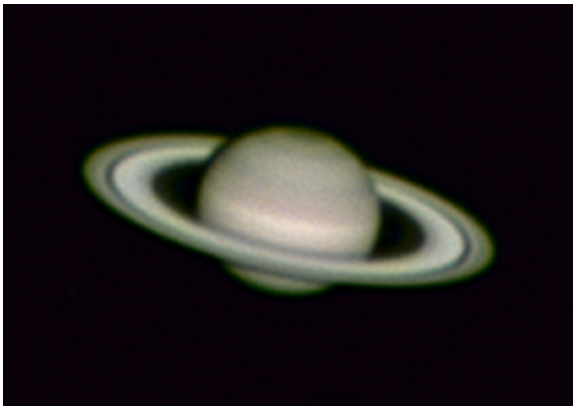


Figura 8.2: Imagen de Saturno obtenida a partir de casi 2000 frames individuales, procesados con Registax 6 y retocadas con Gimp 2.8

Si bien Registax no funciona de manera nativa en Linux, si es posible hacerlo funcionar mediante la utilización de Wine[4].

[4]http://www.winehq.org/

Bibliografía

[1] D. L. Fried, "Probability of getting a lucky short-exposure image through turbulence" J. Opt. Soc. Am. 68, 1651-1657 (1978)

[2] MT9M001 - 1/2-Inch 1.2MP Digital Image Sensor, Rev F (05/2006)

[3] MT9T001 - 1/2-Inch 3.1MP Digital Image Sensor Datasheet, Rev D (07/2005)

[4] CY7C68013A High-Speed USB Peripheral Controller Datasheet, Rev V (02/2012)

[5] Image Sensor Architectures for Digital Cinematography - DALSA Digital Cinema. Pág 8.

[6] Understanding Digital Camera Sensors - `http://www.cambridgeincolour.com/tutorials/camera-sensors.htm`. Visitado el 23 de enero de 2014.

[7] ST-4 Star Tracker Imaging Camera - Operation Manual.

[8] Debayering Demystified, Craig Stark.

[9] The Netpbm Formats - `http://netpbm.sourceforge.net/doc/#formats`. Visitado el 11 de enero de 2014.

[10] The PBM Format - `http://netpbm.sourceforge.net/doc/pbm.html`. Visitado el 11 de enero de 2014.

[11] PGM Format Specification - `http://netpbm.sourceforge.net/doc/pgm.html`. Visitado el 11 de enero de 2014.

[12] PPM Format Format Specification - `http://netpbm.sourceforge.net/doc/ppm.html`. Visitado el 11 de enero de 2014.

[13] Lanlan Chang, Yap-Peng Tan, "Hybrid color filter array demosaicking for effective artifact suppression", J. Electron. Imaging. 15(1), 013003 (2006).

[14] Henrique S. Malvar, Li-wei He, and Ross Cutler, "High-quality Linear Interpolation for demosaicing of bayer-parrerned color images", Microsoft Research, IEEE (2004)

[15] QhyCCD Linux World. `http://qhyccd.com/ccdbbs/index.php?board=16.0`.

[16] Mark J. Coco, "Guiding a Telescope for Imaging", Sky and Telescope `http://www.skyandtelescope.com/howto/astrophotography/3304266.html`. Visitado el 23 de marzo de 2014

[17] D. Peach's Views of the Solar System. `http://www.damianpeach.com/`

[18] Astrophotography from Cebu City, Philippines by Christopher Go. `http://astro.christone.net/`

Agradecimientos

A Hongyun Qiu quién permitió el acceso a los fuentes del driver para Windows. A Tom Van den Eede, desarrollador del driver para Windows para la cámara QHY5T, su ayuda fue fundamental durante la última etapa de desarrollo del controlador. A Geoffrey Hausheer quien hizo disponible su controlador para la cámara QHY5 bajo una licencia Libre. A Daniel Holler, Andrew Stepanenko, Henrik Kressner y demás miembros de la comunidad de usuarios QhyCCD-GNU/Linux quienes contribuyeron ideas y código al trabajo aquí presentado. A Pavol Ďuriš y Ioannis Ioannou quieres ayudaron a probar el driver y el visor. A Giampiero Spezzano y Clive Rogers por no permitir que me rindiera en las horas más oscuras y por el código e ideas que aportaron. A Andoni Zubimendi por su acertada observación respecto al firmware y la ayuda con el análisis. A Gastón Traberg por su ayuda con el programa para extraer el firmware. A Emanuel Borda por el interés y la ayuda brindada durante las primeras etapas del proyecto. A Beatriz García, Belén Iazzetta, Aldana Gómez Ríos y Lautaro De León, por las observaciones al texto de la tesina. A mi directora, Lía Molinari por su entusiasmo.